Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática,
2016

Mário Jorge Rodrigues Pina

Monitorização de Desempenho e Previsão de Falhas em Servidores Cloud

universidade de aveiro

Universidade de Aveiro Departamento de Electrónica, Telecomunicações e Informática,
2016

Mário Jorge
Rodrigues Pina

Monitorização de Desempenho e Previsão de Falhas em Servidores Cloud

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Professor Doutor Paulo Jorge Salvador Serra Ferreira, Professor Auxiliar do Departamento Instituto de Telecomunicações da Universidade de Aveiro.

o júri / the jury

presidente / president

Professor Doutor André Ventura da Cruz Marnoto Zúquete
Professor Auxiliar, Universidade de Aveiro (por delegação da Reitora da Universidade de Aveiro)

vogais / examiners committee

Professor(a) Doutor(a) Rui Jorge Morais Tomaz Valadas
Professor Catedrático, Universidade Técnica de Lisboa (Arguente Principal)

Professor Doutor Paulo Jorge Salvador Serra Ferreira
Professor Auxiliar, Universidade de Aveiro (Orientador)

Agradecimentos

O meu maior e mais profundo agradecimento ao meu orientador, Professor Paulo Jorge Salvador Serra Ferreira, por todo o apoio, disponibilidade e orientação dado ao longo de todo o desenvolvimento desta dissertação.

Agradeço também aos meus pais, não só pelas oportunidades que me permitiram usufruir nestes últimos cinco anos, mas também por todo o apoio que me deram ao longo de todo o meu percurso académico e mesmo fora dele.

Deixo também uma palavra de apreço a toda a minha família, por todos os conselhos e momentos passados ao longo de todos estes anos.

E por fim, gostaria de agradecer à minha namorada e a todos os meus amigos por toda a paciência e ajuda nesta fase final, e principalmente pela amizade com que me presenteiam todos os dias desde que entrei para a Universidade.

A todas estas pessoas gostaria de deixar o meu mais sincero obrigado.

Resumo

Nos dias atuais o uso de *cloud servers* tem aumentado devido às diversas vantagens que estes oferecem em relação ao uso de servidores físicos. Os servidores físicos, ao estarem dependentes de um único dispositivo de *hardware*, sofrem de um único ponto de falha, enquanto os servidores *cloud* são entidades de *software* independente das unidades de *hardware*, e portanto, não possuem pontos de falha. Os servidores *cloud* permitem, além disso, uma melhor otimização dos custos de operação, ao possibilitar uma alocação variável dos recursos ao longo do tempo, conforme as necessidades.

No entanto, em diversas situações, existem fornecedores de servidores que tiram proveito do facto dos recursos alocados aos servidores raramente serem utilizados a 100% pelos seus clientes. Desta forma, é possível que uma infraestrutura de servidores tenha capacidade para um certo número de clientes, mas o número efetivo de clientes ativos seja superior. A partilha de recursos na infraestrutura implica que a capacidade de um cliente aceder à totalidade dos recursos a si alocados, depende fortemente do uso dos recursos alocados a outros clientes.

Esta dissertação teve como objetivos: (i) a criação e desenvolvimento de um sistema de monitorização da disponibilidade de recursos computacionais num servidor *cloud*, e (ii) desenvolvimento de metodologias de previsão de indisponibilidade de recursos (falhas) a curto e médio prazo.

Com este sistema é possível otimizar o desempenho coletivo de um grupo de servidores *cloud*, adaptando o comportamento individual de cada servidor *cloud* à eventual redução de recursos computacionais. A adaptação comportamental de cada servidor pode passar: (i) pela delegação de tarefas importantes em outros servidores do grupo com recursos disponíveis e/ou (ii) pelo adiamento de tarefas menos importantes, até que haja disponibilidade de recursos.

Abstract

Nowadays, the use of cloud servers has been rising due to the advantages that they offer when compared with physical servers. Physical servers depending only on one hardware device have a single point of failure, while cloud servers are hardware independent software units, and therefore, do not have tha liability. Moreover, cloud servers allow the optimization of operational costs by providing variable computational resources allocation over time, according to operational needs.

However, on many occasions server providers take advantage of the fact that allocated resources are rarely used at 100% capacity by its clients. This way, it is possible that a cloud server infrastructure have the capacity for a certain number of clients, but the real number of active clients is higher. Resources sharing implies that the ability for a client to access the overall capacity of the resources allocated to him, depends greatly on the usage of resources by other clients.

This dissertation has two main objectives: (i) the creation and development of a system for performance monitoring in a cloud server, and (ii) development of methodologies to predict resources starvation (failures) at short- and medium-term.

With this system it is possible to optimize the overall performance of a group of cloud servers, adapting the individual behavior of each cloud server to eventual resource starvation events. The behavioral adaptation of a server may consist in: (ii) delegating important tasks to other group cloud servers with available resources and/or (ii) delaying low importance tasks until availability of resources.

Conteúdo

Lista de Figuras

Lista de Tabelas

1 | Introdução

Nesta dissertação é apresentado todo o trabalho de pesquisa e desenvolvimento necessário para a implementação de um sistema de monitorização da disponibilidade de recursos computacionais e previsão da indisponibilidade de recursos (falhas) a curto e médio prazo em servidores *cloud*. Este trabalho centrou-se principalmente na previsão do estado de ocupação do processador, assim como do estado de ocupação da memória *RAM* e da rede do servidor.

1.1 Motivo e Objetivos

Com o aumento da capacidade de processamento e desempenho da tecnologia o uso de servidores *cloud* tem aumentado em empresas, instituições, universidades, entre outros. Isto deve-se ao facto destes terem um custo de investimento e operação muito mais reduzido que servidores físicos, em resultado da otimização dos recursos instalados pela partilha dos mesmo por múltiplos utilizadores. O menor custo e facilidade de aquisição de um servidor *cloud*, possibilita a criação de infraestruturas distribuídas, o que por usa vez, permite a otimização de serviços em termos de desempenho e fiabilidade.

Os servidores *cloud*, ao serem independentes das unidades de *hardware*, não sofrem de um único ponto de falha, criando uma maior segurança e redundância dos dados. São também extremamente maleáveis, podendo ser aumentada ou diminuída a capacidade computacional conforme a vontade do cliente, permitindo às empresas e diversas instituições uma melhor gestão de recursos computacionais e monetários. No entanto, empresas que proporcionam este tipo de serviços, podem tirar proveito do facto dos recursos alocados aos servidores raramente serem utilizados a 100% pelos seus clientes. Isto faz com que seja possível numa infraestrutura de servidores com capacidade para um determinado número de clientes, estejam na verdade um número superior de clientes, tendo alocado mais recursos virtuais do que aqueles que o servidor tem na realidade. Esta sobrelotação de recursos pode levar a que exista um maior pedido de acessos aos recursos do sistema, fazendo com que os servidores percam alguma da sua performance. Isto pode criar problemas aos clientes que os utilizam, como perca de informação e desempenho reduzido. No sentido de evitar e prevenir este problema, desenvolveu-se um sistema de monitorização da disponibilidade de recursos e de previsão da indisponibilidade de recursos.

O sistema de monitorização pretende ir verificando o estado de ocupação dos recursos do

servidor no lado do cliente, de forma a que seja possível saber se o recurso está ou não a ser utilizado em demasia, podendo mover as tarefas desse servidor para outro servidor que possua mais recursos disponíveis nesse momento, de forma a evitar os problemas da sobrecarga do sistema. A parte da previsão pretende que, não só seja possível prever uma sobrecarga do sistema antes que esta aconteça, mas também fazer com que a troca de tarefas seja feita sem ser necessária a intervenção do cliente, ajudando assim ao aumento do desempenho e produtividade de formas distintas.

O trabalho realizado no âmbito desta dissertação foi realizado em paralelo com o aluno José Luís da Silva Rosa no âmbito da sua dissertação de mestrado *Customer-Side Detection of BGP Routing Attacks*[1] apresentado no ano letivo 2015/2016, onde ao servidor desenvolvido nessa dissertação, foi integrado o sistema de previsão da indisponibilidade de recursos do servidor realizado nesta dissertação.

1.2 Estrutura da dissertação

Esta dissertação encontra-se dividida em seis capítulos, seguindo a seguinte estrutura:

- **Introdução -** Neste capítulo é descrito o problema que se pretende resolver, assim como uma breve descrição da solução desenvolvida;
- **Estado da Arte -** Neste capítulo é feita uma introdução de vários contextos e definições necessárias para a compreensão do problema e da solução.
- **Monitorização da Disponibilidade de Recursos -** Neste capítulo é descrito todo o estudo efetuado de forma a se provar que é possível realizar-se uma monitorização de desempenho de um servidor, através dos seus clientes.
- **Previsão da Indisponibilidade de Recursos -** Neste capítulo é descrito todo o estudo efetuado de forma a se provar que é possível realizar-se uma previsão de desempenho de um servidor, através da monitorização realizada pelos seus clientes.
- **Descrição do Sistema -** Neste capítulo é feita uma descrição da solução implementada e das suas componentes principais.
- **Conclusão -** Neste capítulo é feita uma breve conclusão, analisando o trabalho efetuado, assim como algumas últimas considerações para trabalhos futuros.

2 | Estado da Arte

Irá ser demonstrado o estudo inicial sobre os diferentes tipos de *cloud servers* e são também apresentadas propostas para a criação de diferentes tipos de *benchmark* para a análise da performance dos diferentes servidores.

2.1 *Cloud Servers*

Cloud server é um servidor virtual que corre num ambiente de *cloud computing* (tipo de computação baseado na internet onde recursos, dados e informação partilhados são disponibilizados a computadores e outros dispositivos).[2]

Quando um cliente opta por um *cloud server*, este está a alugar espaço virtual, em vez de alugar (ou comprar) servidores físicos. Tradicionalmente existem duas opções:

- ***Shared hosting:*** a opção mais barata, onde os servidores são partilhados entre diversos clientes. O *website* de um cliente poderá ser hospedado no mesmo servidor que os *websites* de outros clientes. Isto trás algumas desvantagens, incluindo um aumento no tráfico de acesso ao servidor, e uma configuração mais inflexível.

- ***Dedicated hosting:*** opção mais cara, onde cada cliente aluga servidores inteiros. Isto significa que não há necessidade de partilha com outros clientes e permitem um controlo mais completo do servidor.

Através do uso de *cloud servers*, os clientes conseguem fazer uma melhor gestão dos seus recursos (e dos seus custos associados), uma vez que pode aumentar ou diminuir os seus recursos com uma maior facilidade. Permitem também que vários servidores sejam atribuídos ao mesmo cliente, o que adiciona uma capacidade de redundância e de tolerância a falhas, uma vez que caso um servidor falhe, outros podem tomar o seu lugar.[3]

Cloud servers podem correr em *virtual machine monitors*.

2.1.1 *Cloud Servers vs Virtual Private Servers*

A principal diferença entre *Virtual Private Servers* (*VPS*) e *cloud servers* é a escalabilidade. Os *VPS* têm sido reconhecidos como uma forma de reduzir custos e aumentar a eficiência computacional de empresas. Ao isolar aplicações e programas num único servidor virtual fornecem um alto nível de privacidade, segurança e controlo. No entanto, faltam-lhes a capacidade de escalabilidade.

Um *cloud server* permite uma grande escalabilidade do sistema (quer seja aumentar ou diminuir), alterando facilmente os recursos do servidor, aumentando também a facilidade com que se possam fazer copias do servidor para *back-up*.[4] Devido a isso, a tolerância a falhas nos *cloud servers* é muito mais elevada do que em *VPS*. Um *cloud server* permite também realizar todo o tipo de virtualização. Não só de recursos internos (como *RAM*, *CPU*, memória, etc), como também de recursos externos (rede).

2.2 *Hypervisor*

Um *hypervisor* ou *virtual machine monitor* (*VMM*) é um programa que permite que um único computador suporte múltiplos ambientes de execução idênticos. Todos os utilizadores veem o seu sistema como um computador isolado dos restantes utilizadores, apesar de cada utilizador ser servido pela mesma máquina. Neste contexto, uma máquina virtual (uma réplica isolada e eficiente de uma maquina real) é um sistema operativo que é controlado por um programa de controlo subjacente.[5]

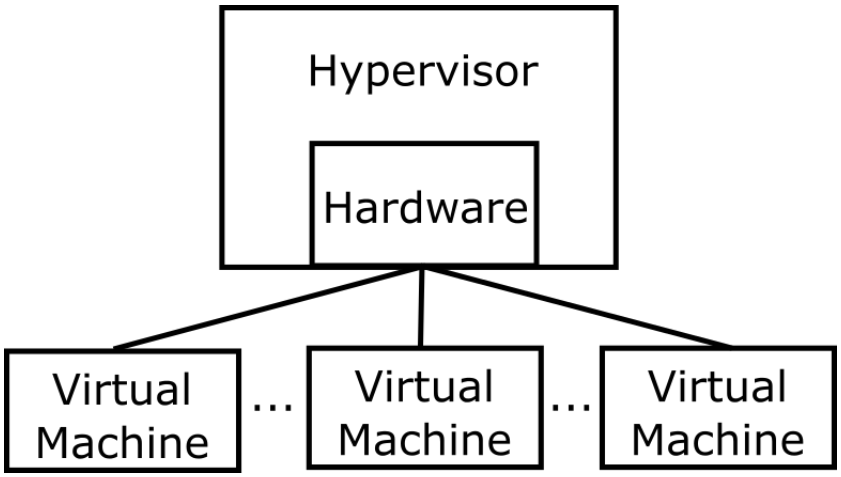


Figura 2.1: Representação de um *hypervisor*.

Como peça de *software* um *hypervisors* tem três caraterísticas principais:

1. O *hypervisors* fornece um ambiente para programas que é essencialmente idêntico à máquina original;

2. Programas que corram neste ambiente mostram, no pior dos casos, pequenas discrepâncias no tempo de execução;

3. O *hypervisors* tem controlo completo dos recursos do sistema.

Por ambiente "essencialmente idêntico", a primeira característica, entende-se o seguinte: qualquer programa que corra sob um *hypervisor*, deve demonstrar uma comportamento idêntico ao demonstrado caso o programa fosse executado na máquina original diretamente, com a possibilidade de algumas diferenças excecionais, causadas pela disponibilidade dos recursos do sistema e diferenças causadas por dependências de tempo.

A segunda característica é a eficiência. Exige que um subconjunto estatisticamente dominante das instruções do processador virtual sejam diretamente executados pelo processador real, sem interferência por parte do *hypervisor*.

A terceira caraterística, controlo de recursos, refere-se como recursos aos itens usuais tais como memória, periféricos e semelhantes, apesar de não incluir necessariamente as atividades do processador. Diz-se que o *hypervisors* tem o controlo completo destes recursos se:

- Não for possível a um programa em execução, aceder a recursos que não lhe foram explicitamente concedidos;
- É possível a qualquer altura, sob certas circunstâncias, que o *hypervisor* recupere o controlo de recursos previamente concedidos.[6]

2.2.1 Classificação

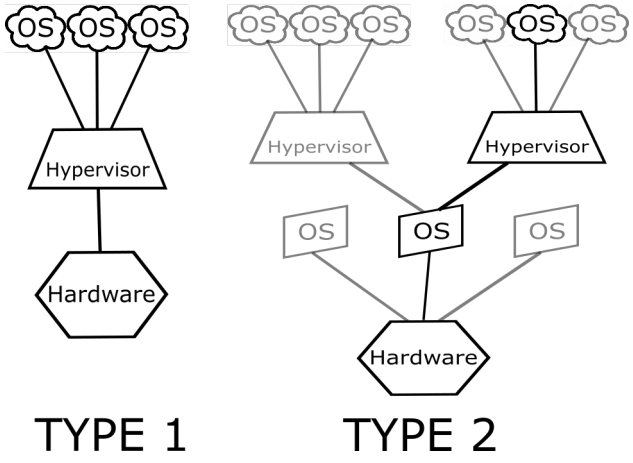


Figura 2.2: Tipos de Classificação de *hypervisors*.

Os *hypervisors* podem ser classificados em dois tipos:

1. **Tipo 1: nativo ou bare-metal.** Estes *hypervisors* correm diretamente no hardware

do servidor para controlar e gerir os sistemas operativos dos clientes. Um cliente corre como um processo no hospedeiro.

2. **Tipo 2: hospedado.** Estes *hypervisors* correm num sistema operativo convencional como um outro programa computacional. Os sistemas operativos dos clientes funcionam "em cima" do sistema operativo hospedeiro.[6]

2.2.2 Virtualização de rede

Virtualização de rede é o processo de combinar *hardware* e *software* dos recursos de rede numa única entidade de *software* administrativa, dividindo a largura de banda da rede em diferentes canais, independentes entre si, que podem ser atribuídos (e reatribuídos) a um servidor particular em tempo real.

Esta virtualização pode ser dividida em duas categorias:

- **Virtualização externa:** combina diversas redes, ou partes de redes, numa única unidade virtual, de forma a melhorar a sua eficiência. Uma rede virtual *VLAN*) e um *switch* de rede são as componentes principais. Através desta tecnologia, o administrador de rede pode configurar sistemas que estejam ligados á mesma rede física, criando diversas redes virtuais distintas.
- **Virtualização interna:** fornece funcionalidades de rede aos contentores de software num único servidor de rede. Isto pode melhorar a eficiência de um sistema ao isolar aplicações em contentores distintos.

A ideia da virtualização é dividir uma rede complexa em diferentes partes facilmente geridas por um administrador. Pretende-se com isto otimizar a velocidade, segurança, escalabilidade e flexibilidade da rede. A virtualização é particularmente útil quando uma certa rede recebe um aumento súbito de utilização.[7]

2.2.3 OpenVZ

OpenVZ é uma virtualização de sistema operativo para o *Linux*. O *OpenVZ* cria diversos *VPS* (*Virtual Private Servers* – também chamados de *virtual environments* (*VE*)) seguros e isolados num único servidor físico fornecendo uma melhor utilização do servidor e garantindo que as aplicações não entrem em conflito.[8] Todos os *VPS* partilham a mesma arquitetura e versão de *kernel*, pois o *OpenVZ* utiliza um único *patch* do *Linux Kernel*, forcando todos os *VPS* a utilizar o mesmo sistema operativo. Esta abordagem pode trazer desvantagens caso os utilizadores precisem de versões diferentes que não a do hospedeiro. No entanto, como não tem a sobrecarga de um *hypervisor* verdadeiro, é bastante rápido e eficiente.

A alocação de memória é suave. A memória que não está a ser utilizada por um *VPS* pode ser utilizada por outros *VPS* ou pela *cache* do disco. Enquanto versões mais antigas

do *OpenVZ* utilizavam um sistema de ficheiros comum. Cada *VPS* era apenas um diretório isolado utilizando *chroot*. Esta operação em sistemas operativos *Linux* altera o diretório *root* aparente do processo atual e dos seus descendentes. Neste ambiente de virtualização um programa não pode aceder a ficheiros localizados fora do seu diretório atribuído. Atualmente existem já versões que permitem a cada *VPS* ter o seu próprio sistema de ficheiros.[9]

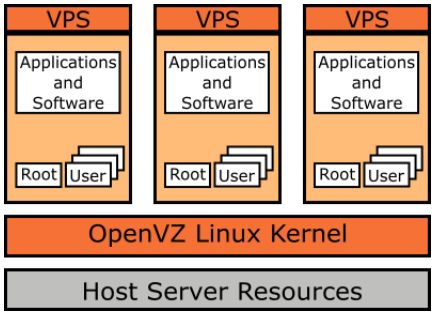


Figura 2.3: Arquitetura do *OpenVZ*.

A virtualização de rede do *OpenVZ* é desenhada de forma a isolar *VPS* uns dos outros e da rede física:

- Cada *VPS* tem o seu próprio endereço IP. Múltiplos IPs por *VPS* é permitido.
- O tráfego de rede de um *VPS* é isolado dos restantes *VPS*. Ou seja, os *VPS* estão protegidos uns dos outros para que *snooping* do tráfego seja impossível.
- *Firewalls* podem ser utilizadas dentro de um *VPS*. Ou seja é possível configurar uma *firewall* através do próprio *VPS*.
- É possível fazer a manutenção das tabelas de encaminhamento e funcionalidades de encaminhamento avançadas por *VPS* individuais. Por exemplo, definir diferentes *maximum transmission units* (*MTU's*) para destinos distintos, especificar diferentes endereços de origem para diferentes destinos, etc.[10]

2.2.4 Xen

Xen é um *hypervisor* do tipo 1, correndo num estado mais privilegiado da *CPU* que o restante software. O *Xen* é gerido por uma conta privilegiada a correr no *hypervisor* denominada por *Domain-0* (*Dom0*). *Dom0* é um *Linux kernel* especialmente modificado que é iniciado pelo *Xen* durante o início do sistema. É responsável por gerir todos os aspetos das restantes não privilegiadas máquinas virtuais (*Domais-Us* (*DomUs*)) que correm também no *hypervisor*.

Podem existir um, ou mais, *DomUs* a correr em simultâneo, no entanto nenhum deles tem acesso direto ao hardware físico. Em vez disso têm que fazer pedidos de acesso ao CPU, I/O e acessos ao disco ao *Xen hypervisor*.[11]

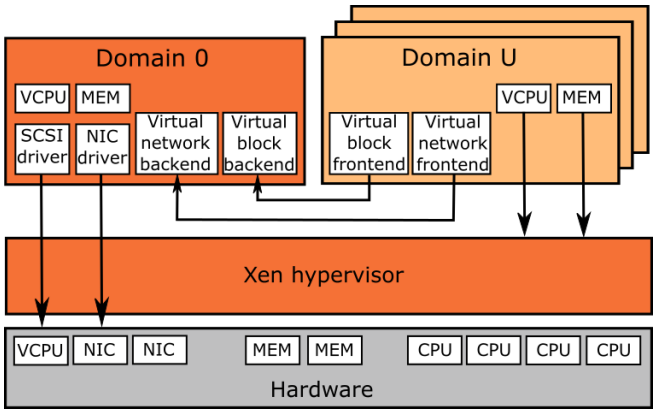


Figura 2.4: Arquitetura do *Xen*.

Na virtualização de rede do *Xen*, cada utilizador tem usualmente acesso a uma ou mais interfaces de rede. Estas interfaces, permitem uma comunicação de rede rápida e eficiente para vários domínios sem a necessidade de criar um emulador de um dispositivo de rede real. Esta virtualização consiste num par de dispositivos de rede. O primeiro (*frontend*) irá residir na conta do utilizador, enquanto o segundo (*backend*) irá residir no domínio de *backend* (usualmente *Dom0*). Para cada utilizador é criado um par semelhante de dispositivos. Este par de dispositivos estão conectados através de um canal virtual de comunicação. É dado acesso a uma conta de utilizador, ao fazer com que o tráfego de rede passe desde o dispositivo de *backend* até á rede física.[12]

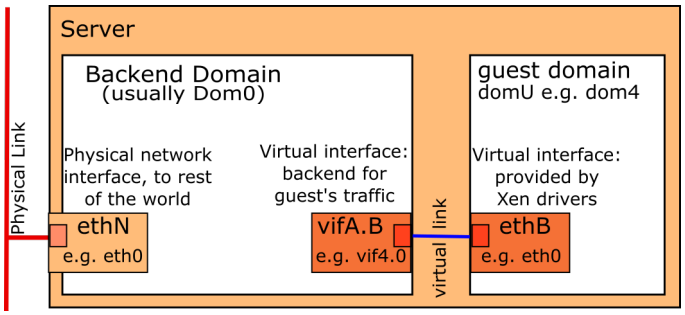


Figura 2.5: Virtualização de rede do *Xen*.

2.2.5 Kernel-based Virtual Machine

Kernel-based Virtual Machine (*KVM*) é implementado como um módulo *kernel* que, quando carregado, converte o *kernel* num *hypervisor* de tipo 1. Ao converter o *Linux kernel* num *KVM hypervisor*, os programadores podem tirar vantagem de várias componentes já existentes no *kernel*, tal como *memory manager* e *scheduler*, em alternativa a terem que os construir do zero.

A arquitetura do *KVM* permite que, do ponto de vista do hospedeiro, cada máquina virtual seja um processo *Linux* que é gerido, programado e protegido como um processo *Linux*

normal.[13]

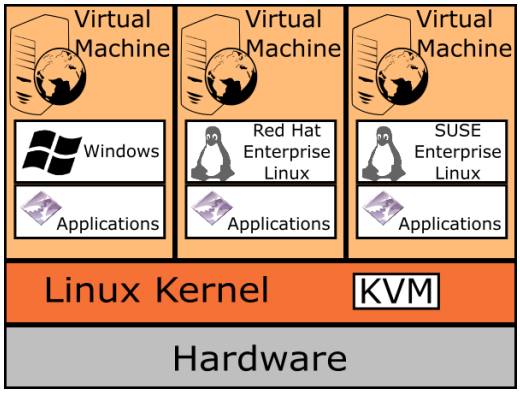


Figura 2.6: Arquitetura do *KVM*.

O *KVM* fornece a cada *VPS* um *Virtualized Network Interface Card* (*VNIC*), que o *VPS* utiliza para realizar todas as suas operações de rede. Mas para realizar comunicação entre dois *VPS* distintos, também fornece um dispositivo virtual (*TUN/TAP*) para cada *VPS*, que está conectada á correspondente *VNIC* do *VPS*, e também a rede física através de uma *bridge virtual* (*VB*). Quando um pacote é recebido na interface física, a *bridge virtual* vai determinar se o pacote é enviado para o *VPS* ou não, e caso seja irá colocar o pacote no *buffer* do dispositivo *TAP*, e o *TAP*, por sua vez, irá copiá-lo para o *VNIC* do *VPS*.[14]

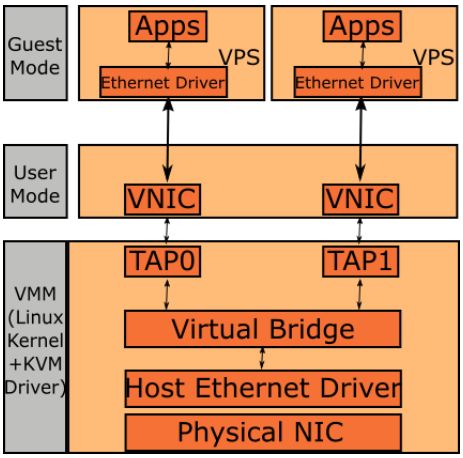


Figura 2.7: Arquitetura de rede do *KVM*.

2.2.6 OpenFlow

OpenFlow é um protocolo de comunicações que permite os utilizadores escolher o percurso dos pacotes, através de uma rede de *switches*. O *OpenFlow* utiliza controladores de rede para determinar os caminhos, sendo que estes controladores são distintos dos próprios *switches*. Isto permite um controlo do tráfego mais sofisticado através do uso de *access control lists* (*ACL's*) e protocolos de *routing*.

Uma das suas principais vantagens é o facto de permitir que *switches* de diferentes empresas (muitas vezes cada uma com a sua própria interface e linguagem de escrita), sejam geridos remotamente através de um único protocolo.

As principais componentes para se poder ter uma rede controlada por esta componente são as seguintes:

- *Switches* capazes de suportar *OpenFlow*;
- Servidor(es) que executem o processo de controlo;
- Base de dados que contém o mapa da rede.

Um *OverFlow switch* consiste numa tabela de encaminhamento contendo várias entradas usadas para realizar pesquisas e encaminhamento e um canal seguro para o controlador, através do qual as mensagens do *OpenFlow* são trocadas entre o *switch* e o controlador.[15]

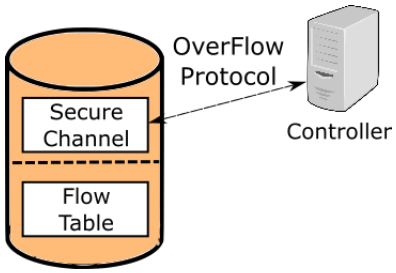


Figura 2.8: *OverFlow*; Relação entre os *switches* e o controlador.

O *OpenFlow* centra-se em obter a lógica da rede numa entidade centralizada, o controlador. O controlador é responsável por todas as decisões de encaminhamento e de *routing* ao gerir e manipular as tabelas de encaminhamento dos *OpenFlow switches*. Quando o tráfego estiver logicamente organizado em fluxos de pacotes, a sua manipulação torna-se bastante mais simples e muito mais direta. Esta característica única do *OpenFlow* pode ser utilizada para obter isolamento do tráfego de rede. Ao agrupar fluxos com características diferentes criam-se partições lógicas de infraestrutura comum da rede física. Ao mapear estes grupos de fluxos em diferentes partições lógicas e guardando estes mapas numa entidade centralizada que contém um mapa completo da rede, é possível criar uma rede virtual baseada em fluxos em cima da rede física.[15]

2.3 Benchmark

Um *benchmark* é o ato de correr um programa, um conjunto de programas, ou uma outra operação, de forma a avaliar a performance relativa de um objeto.

Têm diversa utilidades como, por exemplo, verificar previamente a capacidade de um determinado equipamento de processar um determinado conjunto de operações, e fornecer informações sobre o uso de diferentes implementações de algoritmos.[16]

2.3.1 CPU/Processador Benchmarking

O *benchmark* de *CPU* é utilizado para medir a velocidade a que o sistema é capaz de executar um certo conjunto de instruções.

Existem diversas formas de fazer este tipo de *benchmark*, como por exemplo:

- Cálculo de operações aritméticas complexas;
- Geração de uma lista de números primos;
- Geração da sequência de Fibonacci;
- Operações realizadas em Strings (atribuir novos valores a uma variável).

2.3.2 Memória RAM

O *benchmark* da memória *RAM* mede a capacidade do servidor, em termos de tamanho e/ou tempo de acesso. Regra geral é feito usando blocos contínuos com tamanho na ordem da potência de 2 (16 *MBytes* até 512 *MBytes*, geralmente), e realizando operações de escrita/leitura.

As duas principais formas de se fazer este tipo de *benchmark* são:

- Tentativa de alocação de um número elevado de blocos de pequena dimensão;
- Tentativa de alocação de um número reduzido de blocos de grande dimensão.

Nos casos em que é mais importante saber a velocidade de acesso a blocos na memória do que a velocidade da alocação destes (como por exemplo aceder dentro de um ciclo a um *array* de dados de grandes dimensões), utilizam-se métodos de *benchmark* que executem acessos aleatórios a um bloco de dados de grande dimensão, medindo-se o tempo que demora a obter os dados armazenados.

2.3.3 Rede

O *benchmark* de rede mede a velocidade de transmissão de pacotes pela rede.

Geralmente são enviados pacotes *probe* (como por exemplo, *ICMP Request*) para diversos sites (preferencialmente para sites cujo tempo de vida estimado seja muito elevado (como *Google.com* e *youtube.com*), e medir o tempo até a receção da respetiva resposta (*ICMP reply*)).

No entanto, há que ter em atenção que, certos sites utilizam técnicas contra ataques do tipo *denial of service*. Estas técnicas não permitem que uma única máquina efetue um grande

número de pedidos num curto espaço de tempo, podendo bloquear a ligação. Deste modo os resultados obtidos deste método de *benchmark* são inúteis.

3 | Monitorização da Disponibilidade de Recursos

No caso dos servidores *cloud* os recursos de sistemas são partilhados por diversos clientes, denominados por *virtual private servers* (*VPS*). Por definição um bom *VPS* não pode dar a conhecer ao seu cliente que é um *VPS*, mas sim dar a entender que se trata de um servidor físico dedicado. Devido a isso, a medição dos valores utilizados pelo *hardware* não pode ser feita de forma direta dentro de um *VPS*, uma vez que esses valores apenas nos indicam os valores utilizados pelo próprio *VPS* e não do servidor no qual está alojado. Assim, a forma de fazer monitorização do estado de ocupação dos recursos do sistema tem que ser feita de forma indireta, ao se fazer uma verificação da utilização do recurso que se pretende analisar e fazer uma avaliação dos valores obtidos.

Os principais recursos que maior influência têm na performance dos *VPS* são a utilização do *CPU*, da *memória RAM* e a placa de rede. Por isso, foram estas três componentes onde se concentrou o estudo do desempenho realizado ao longo desta dissertação.

3.1 Análise do tempo para realização de um número fixo de instruções

Uma das possibilidades de se avaliar o uso destas componentes é a medição do tempo que um número fixo de tarefas leva a ser executado. Assim, foi elaborado um programa cujo propósito era o cálculo do número de *fibonacci* de vários valores fixos (no exemplo foram calculados os números na 10ª, 20ª, 30ª, 35ª e 40ª posição), enquanto era feita a medição do tempo que levava a realizar esses cálculos. Em seguida, foram colocadas diversas máquinas virtuais executadas utilizando *KVM*, num único computador. Numa dessas máquinas foram feitas diversas medições do tempo que levava o programa mencionado anteriormente a ser executado. As restantes máquinas virtuais foram utilizadas para variar a ocupação da carga processador.

Os resultados dessas medições foram recolhidos e encontram-se representados na figura 3.1, onde o eixo vertical indica o número de segundos que o mesmo código demorou a correr, e o eixo horizontal representa a passagem do tempo. No gráfico as grandes variações do tempo que o código demora a correr, representam a variação da ocupação do processador. Assim,

este método pode ser utilizado para fazer a análise do seu estado de ocupação. No entanto em casos em que a ocupação do processador era demasiado elevada o programa de monitorização demorava demasiado tempo a correr, chegando mesmo a demorar cerca de oito vezes mais. Como o elevado tempo de execução poderia levar a que o sistema de monitorização viesse a tornar-se parte do mesmo problema que pretende prevenir, esta possibilidade foi excluída e foram feitas estudos numa outra opção.

Figura 3.1: Valores obtidos durante as medições de tempo necessário para a realização de um número fixo de operações.

3.2 Análise do número instruções efetuadas num intervalo de tempo fixo

A possibilidade que foi analisada em seguida foi a medição do número de instruções que o processador conseguir realizar num intervalo fixo de tempo. Desta forma, mesmo que o processador se encontre em sobrecarga, o sistema de monitorização não precisará mais do que apenas uns segundos para fazer a sua análise.

Devido ao sucesso desta nova metodologia, foram criadas várias abordagens para monitorização dos servidores, uma para o processador, uma para a memória *RAM* e outra para a

placa de rede.

3.2.1 Monitorização da disponibilidade de recursos em KVM

Com o intuito de se realizar uma simulação fiável, foi instalado num computador pessoal o sistema de virtualização *KVM*, onde se criaram diversas máquinas virtuais das quais uma foi utilizada para fazer a monitorização da disponibilidade de recursos, e as restantes foram utilizadas para variar a carga dos recursos.

3.2.1.1 Monitorização do *CPU*

Para este caso, tenta-se calcular o valor de π durante um determinado intervalo de tempo. Para esse efeito foi utilizada a formula 3.1.[17]

$$\pi = 4(\sum_{n=0}^{\infty} \frac{-1^n}{2n+1}) \tag{3.1}$$

Assim, o valor dado por esta análise é o número de iterações do somatório $\sum_{n=0}^{\infty} \frac{-1^n}{2n+1}$ que o processador consegue realizar nesse intervalo de tempo. Para este efeito, foi realizado um teste preliminar numa máquina virtual onde foi colocado o programa a correr durante dois segundos. O número de operações executadas durante este teste foi guardado. Enquanto essa análise corria, foram iniciados outros programas de forma a realizar uma ocupação variada do processador, nas restantes máquinas virtuais. Deste teste foram obtidos os resultados da figura 3.2.

No gráfico 3.2 as grandes variações no número de operações, devem-se à variação da ocupação do processador. Assim, podemos verificar que a ocupação do processador influencia o número de operações realizadas, permitindo uma análise mais rápida, sem que o próprio sistema de monitorização se torne parte do problema. Com esta verificação feita, foram realizados mais testes variando o tempo das operações feitas para 2, 5, 22 e 62 segundos de cada vez. Estes testes foram feitos num sistema a correr *KVM*.

Cada teste foi feito em quatro fases, sendo que o sistema possui dois *cores* físicos, e a cada máquina virtual foi disponibilizado um *core* virtual:

1. Uma máquina a executar a verificação de ocupação do *CPU*, **unicamente**;
2. Uma máquina a executar a verificação de ocupação do *CPU*, e **uma** máquina a executar um programa de ocupação de *CPU*;
3. Uma máquina a executar a verificação de ocupação do *CPU*, e **duas** máquinas a executar um programa de ocupação de *CPU*;

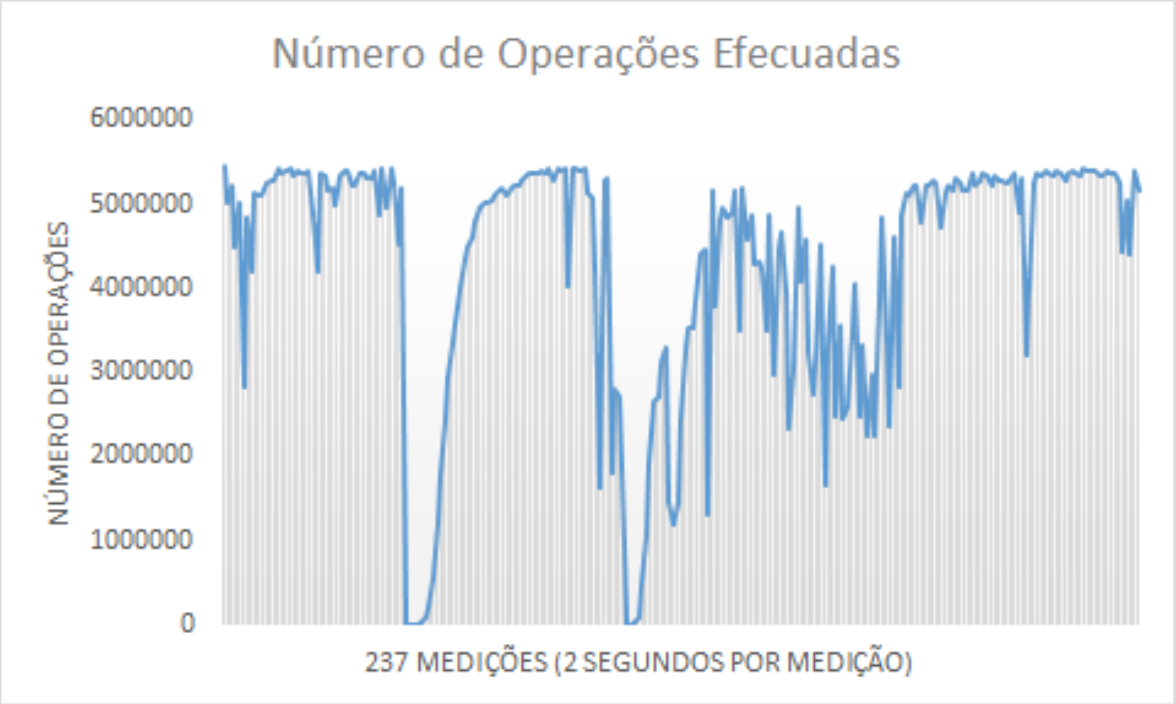


Figura 3.2: Valores obtidos durante a análise da influência da ocupação do processador no número de operações efetuadas num intervalo de tempo fixo.

---

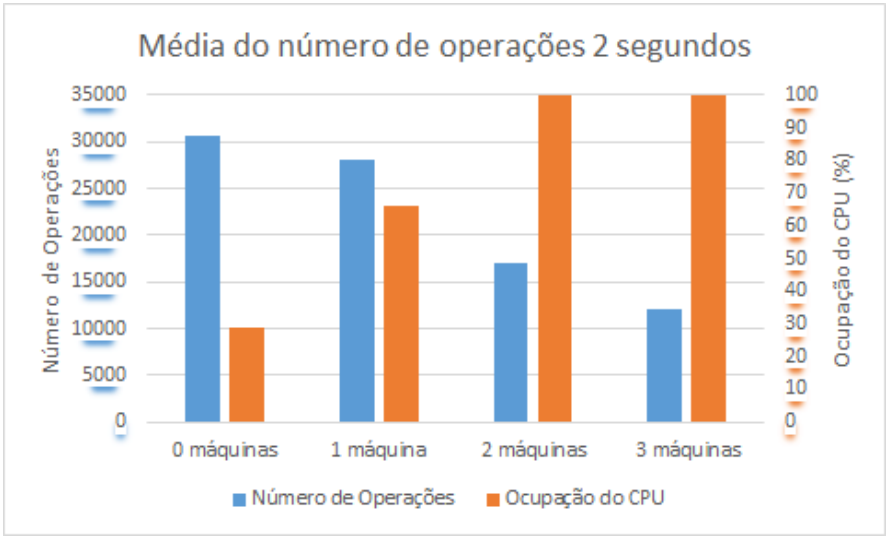
4. Uma máquina a executar a verificação de ocupação do *CPU*, e **três** máquinas a executar um programa de ocupação de *CPU*.

Este método foi aplicado para se verificar o programa em situações de quase nenhuma ocupação de *CPU*, situações com metade da ocupação do *CPU*, situações com a ocupação do *CPU* no limite e situações de sobrecarga da utilização do *CPU*, respetivamente.

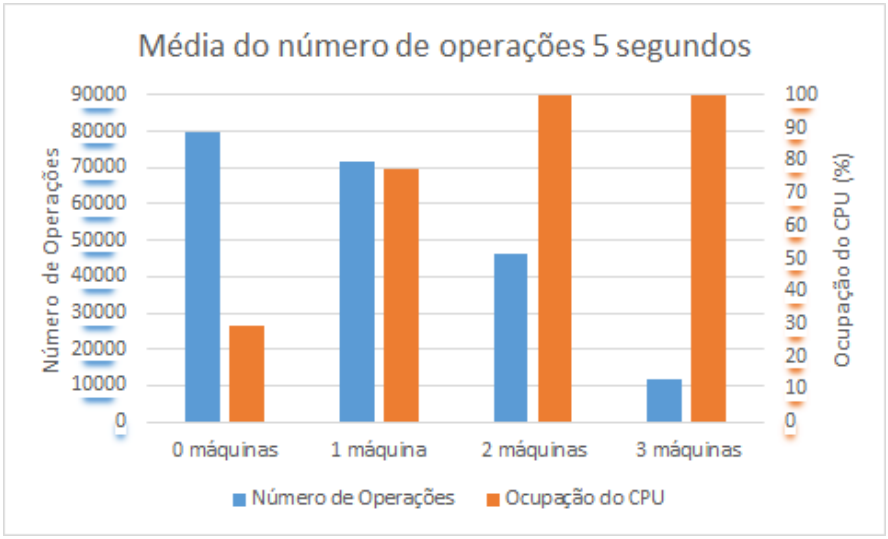
Foi verificado com essa experiência que, apesar de o número de operações realizadas aumentar conforme o período de tempo utilizado (como seria de esperar), existe uma relação inversa entre o número de operações efetuadas pelo programa e a percentagem de ocupação do processador. Ainda podemos concluir que esta relação é independente do tempo de execução do programa (como se pode verificar na imagem 3.3). Assim podemos garantir que, mesmo com um tempo reduzido de medição, é possível fazer uma análise fiável dos resultados.

Um conjunto completo com as imagens dos resultados pode ser consultado no anexo A. Os resultados obtidos na execução de cinco segundos encontram-se na figura 3.4, uma vez que proporciona resultados fiáveis com uma distinção entre os valores mais distinguível, sem a necessidade de um tempo elevado de execução de código, sendo considerado o caso mais favorável para o nosso sistema.

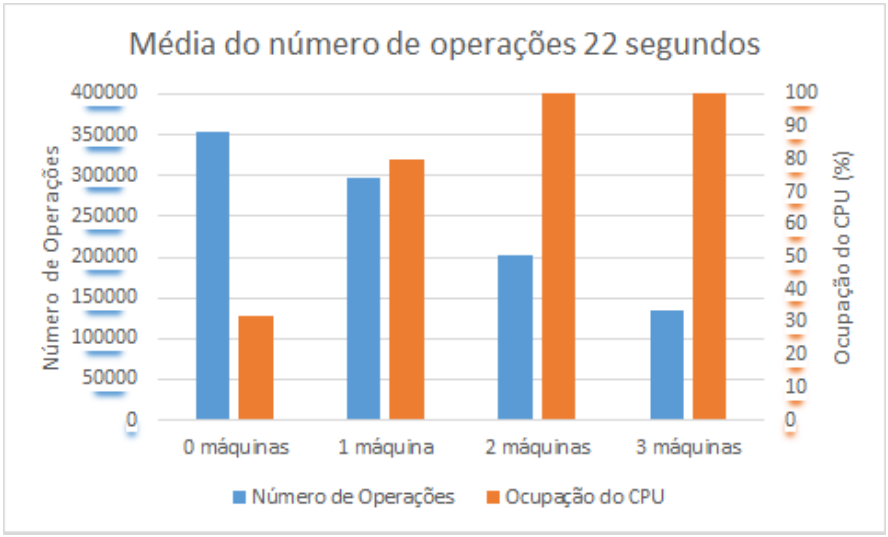
Nas figuras 3.4a e 3.4b podemos observar os valores obtidos diretamente dos testes efetuados, e nas figuras 3.4c e 3.4d encontram-se as médias dos valores, de forma a facilitar a sua



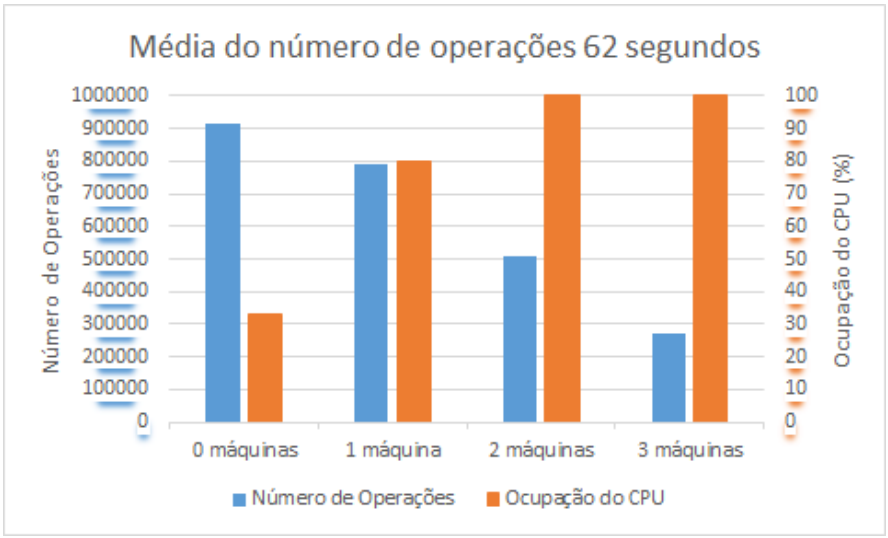
(a) Testes de 2 segundos



(b) Testes de 5 segundos

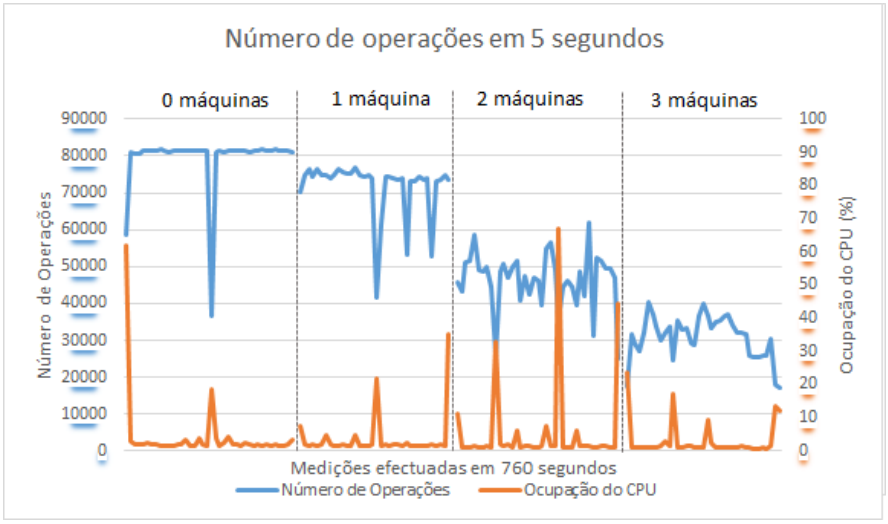


(c) Testes de 22 segundos

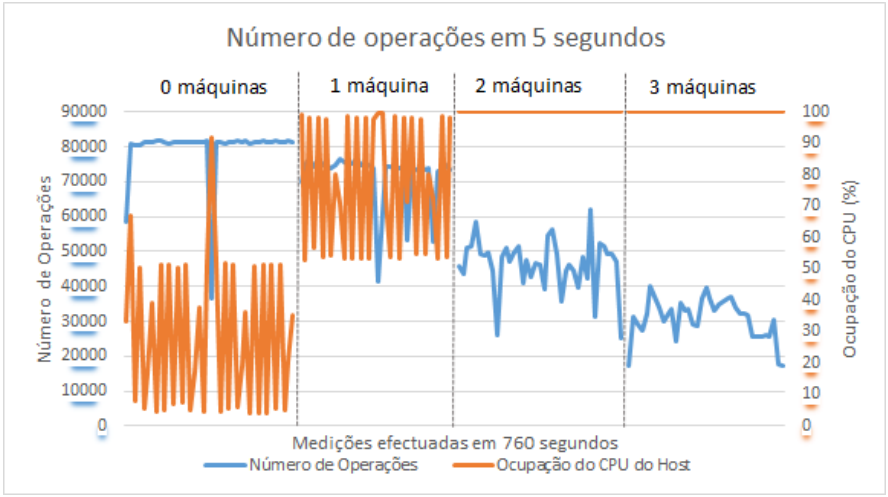


(d) Testes de 62 segundos

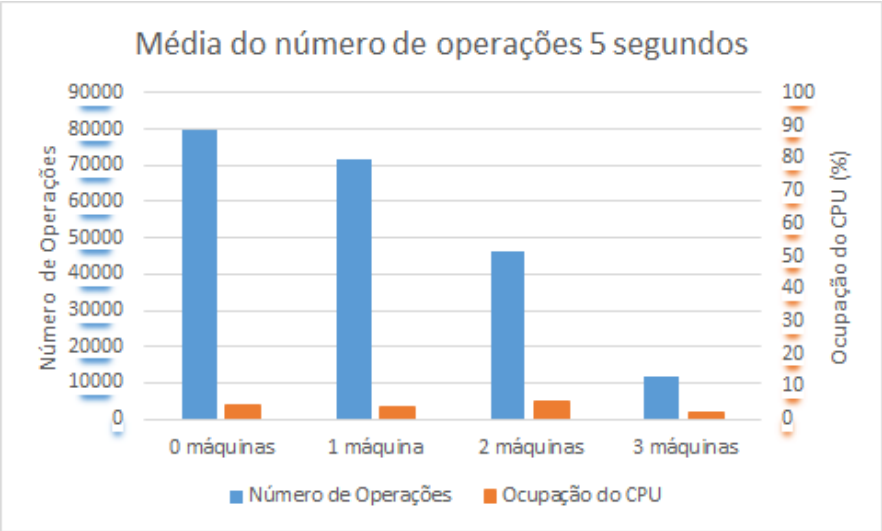
Figura 3.3: Comparação do número médio de Operações efetuadas *vs* Média da ocupação do *CPU* no *Host*, nos diversos tempos de medição de resultados.



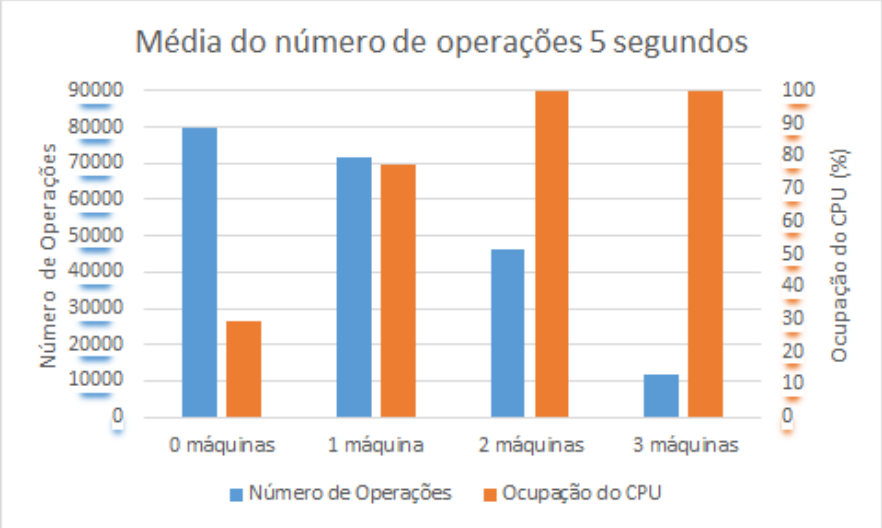
(a) Número de Operações efetuadas vs Medição de ocupação do *CPU* na máquina



(b) Número de Operações efetuadas vs Medição de ocupação do *CPU* no *Host*



(c) Número médio de Operações efetuadas vs Média da ocupação do *CPU* na máquina



(d) Número médio de Operações efetuadas vs Média da ocupação do *CPU* no *Host*

Figura 3.4: Resultados obtidos com a monitorização com intervalo fixo de 5 segundos, variando a ocupação do *CPU*.

observação, assim como demonstrar a similaridade com a monitorização com outros valores de tempo. No anexo A as imagens encontram-se usando o mesmo tipo de ordenação para os restantes valores de monitorização.

Com esta experiência ficou provado que o número de operações realizadas numa máquina virtual a correr num servidor com múltiplos clientes, é influenciado pelo estado de ocupação do processador dos restantes clientes, sendo portanto possível utilizar este métodos para se fazer a monitorização e previsão da indisponibilidade de recursos do servidor por uso excessivo do *CPU*.

3.2.1.2 Monitorização da memória *RAM*

A monitorização da memória *RAM* foi feita de forma semelhante à do *CPU*. A ideia base consiste na contagem do número de operações de ocupação de memória que é possível fazer num intervalo de tempo fixo.

Uma vez que nos computadores/servidores atuais existe um mínimo de memória *RAM* de 128 *MB*, este programa pretende ver o número de alocações de 1 *KB* que a memória consegue alocar no intervalo de tempo pré-definido.

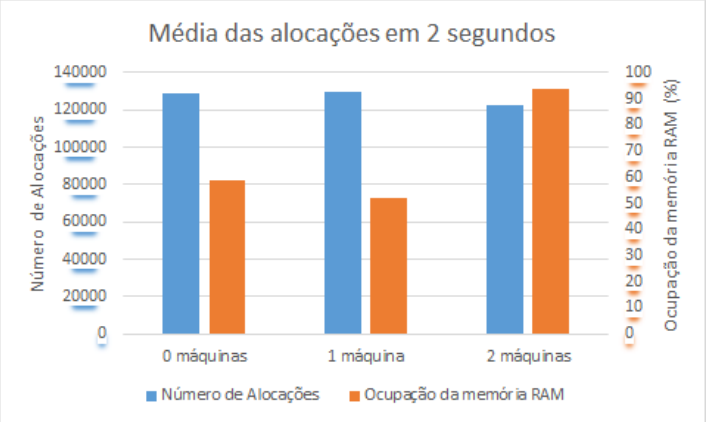
Foram executados testes semelhantes aos realizados para a monitorização de *CPU*, onde se procedeu à verificação da ocupação da memória *RAM* utilizando este novo programa, variando o tempo das operações em 2, 5 e 22 segundos (como o tempo de execução para testes de 62 segundos era demasiado elevado, acabavam por exceder a memória disponível e, portanto, não foram realizados). Estes testes foram feitos num sistema a correr em *KVM*.

Cada teste foi feito em três fases, sendo que o sistema possui 8 *GB* de memória *RAM*, e a cada máquina virtual foram disponibilizados 4 *GB*:

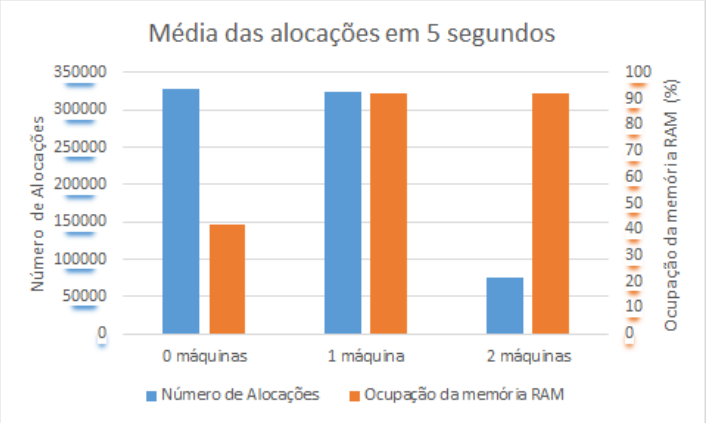
1. Uma máquina a executar a verificação de ocupação da memória *RAM*, **unicamente**;
2. Uma máquina a executar a verificação de ocupação da memória *RAM*, e **uma** máquina a executar um programa de ocupação de memória *RAM*;
3. Uma máquina a executar a verificação de ocupação da memória *RAM*, e **duas** máquinas a executar um programa de ocupação de memória *RAM*;

Este método foi aplicado para estudar o número de alocações do programa, em situações de quase nenhuma ocupação de memória, situações com metade da ocupação da memória e situações com a ocupação da memória no limite, respetivamente.

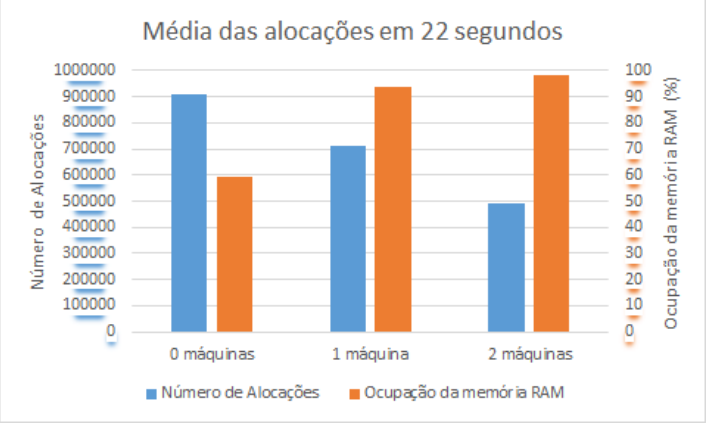
Foi verificado com essa experiência que, apesar de o número de alocações realizadas aumentar conforme o período de tempo utilizado (como seria de esperar), existe uma relação inversa entre o número de alocações efetuadas pelo programa e a percentagem de ocupação da memória *RAM*. Ainda podemos concluir que esta relação é independente do tempo de execução do programa (como se pode verificar na imagem 3.5). Assim podemos garantir que, mesmo com um tempo reduzido de medição, é possível fazer uma análise fiável dos resultados.



(a) Testes de 2 segundos



(b) Testes de 5 segundos



(c) Testes de 22 segundos

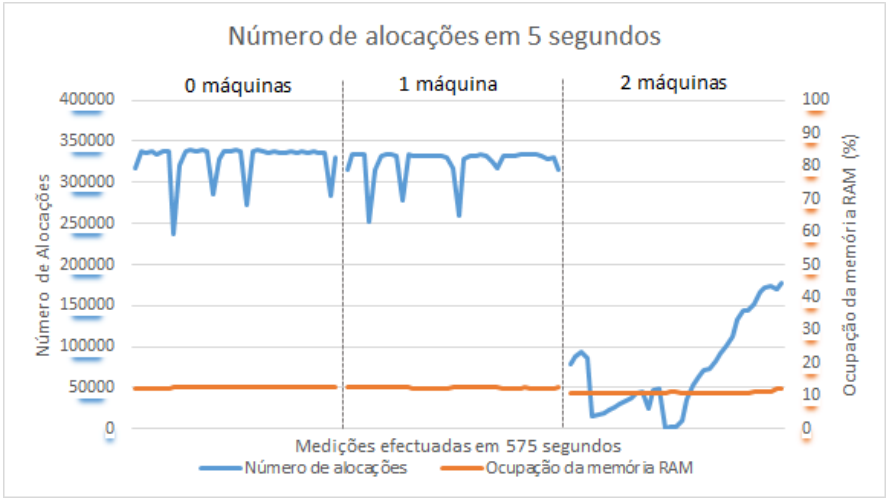
Figura 3.5: Comparação do número médio de alocações efetuadas *vs* Média da ocupação da memória *RAM* no *Host*, nos diversos tempos de medição de resultados.

Um conjunto completo com as imagens dos resultados pode ser consultado no anexo B. Os resultados obtidos nos testes de 5 segundos encontram-se na figura 3.6, uma vez que proporciona resultados fiáveis e com uma distinção entre os valores distinguível, sem a necessidade de um tempo elevado de execução de código, sendo considerado o caso mais favorável para o nosso sistema. .

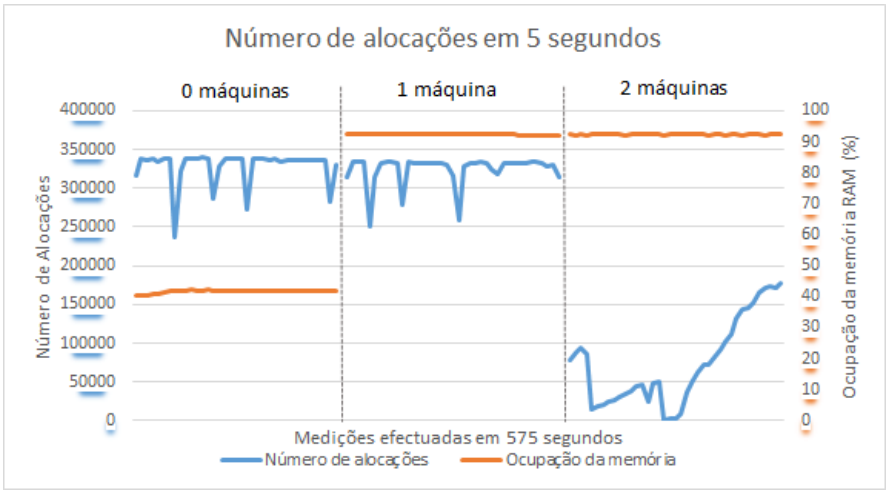
Nas figuras 3.6a e 3.6b podemos observar os valores obtidos diretamente dos testes efetuados, e nas figuras 3.6c e 3.6d encontram-se os valores das médias dos valores, de forma a facilitar a sua observação, assim como demonstrar a similaridade com a monitorização com outros valores de tempo. No anexo B encontram-se as figuras dos restantes tempos de monitorização, utilizando o mesmo tipo de ordenação.

Com esta experiência ficou provado que o número de alocações realizadas numa máquina virtual a correr num servidor com múltiplos clientes, é influenciado pelo estado de ocupação da memória *RAM* dos restantes clientes, sendo portanto possível usar este métodos para se fazer a monitorização e previsão da indisponibilidade de recursos do servidor por uso excessivo da memória *RAM*.

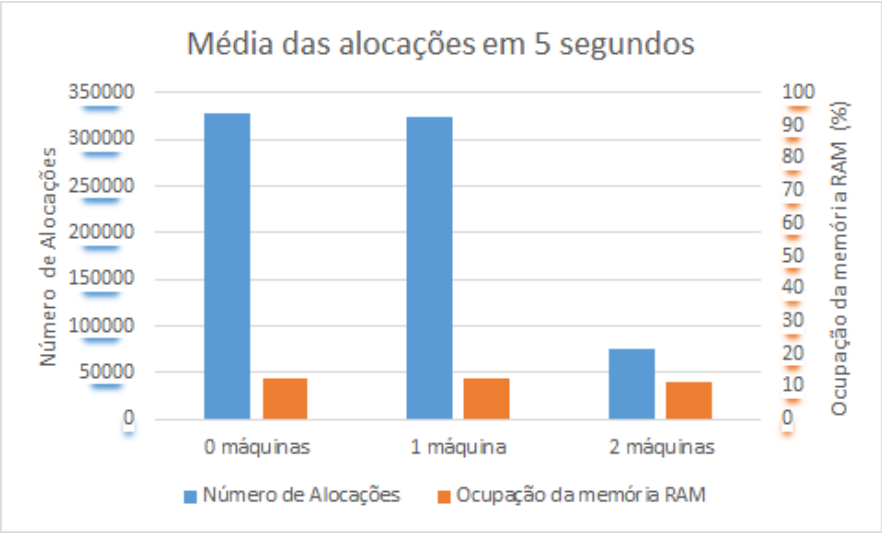
Esta relação é bastante mais acentuada do que no caso da ocupação do *CPU*, sendo que alterações elevadas no número de alocações pode significar que o sistema já se encontra em estado crítico da ocupação da memória RAM.



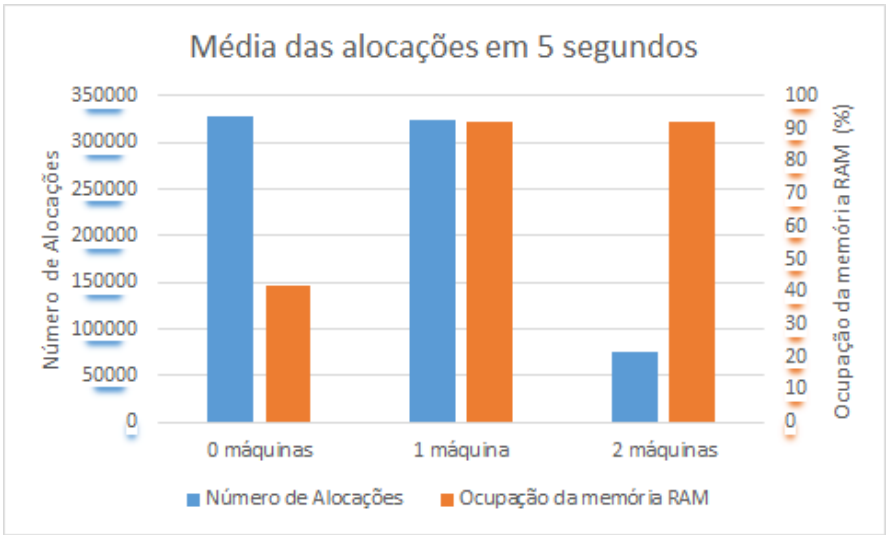
(a) Número de alocações efetuadas vs Medição de ocupação da memória *RAM* na máquina



(b) Número de alocações efetuadas vs Medição de ocupação da memória *RAM* no *Host*



(c) Número médio de alocações efetuadas vs Média da ocupação da memória *RAM* na máquina



(d) Número médio de alocações efetuadas vs Média da ocupação da memória *RAM* no *Host*

Figura 3.6: Resultados obtidos com a monitorização com intervalo fixo de 5 segundos, variando a ocupação a memória *RAM*.

3.2.1.3 Monitorização de rede

A monitorização da rede é feita através do envio de pacotes para diferentes servidores, em diferentes zonas geográficas. Para efeitos de monitorização foram feitos testes para se obter o número de pares de pacotes *ICMP request* e *ICMP reply* que eram possíveis realizar num intervalo de tempo fixo. Ao mesmo tempo que se fazia esta monitorização, variou-se o número de máquinas que usavam continuamente a placa de rede.

Os resultados obtidos referem-se a tempos de execução de 2, 5 e 22 segundos. Estes testes foram feitos num sistema a correr em *KVM*.

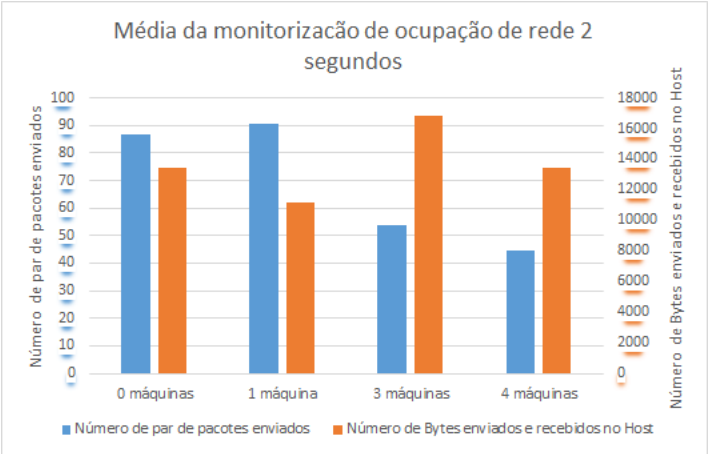
Cada teste foi feito nas cinco fases seguintes:

1. Uma máquina a executar a verificação de ocupação da rede, **unicamente**;
2. Uma máquina a executar a verificação de ocupação da rede, e **uma** máquina a efetuar pedidos *ping ICMP* de forma contínua para um endereço distinto;
3. Uma máquina a executar a verificação de ocupação da rede, e **duas** máquinas a efetuar pedidos *ping ICMP* de forma contínua para um endereço distinto;
4. Uma máquina a executar a verificação de ocupação da rede, e **três** máquinas a efetuar pedidos *ping ICMP* de forma contínua para um endereço distinto;
5. Uma máquina a executar a verificação de ocupação da rede, e **quatro** máquinas a efetuar pedidos *ping ICMP* de forma contínua para um endereço distinto;

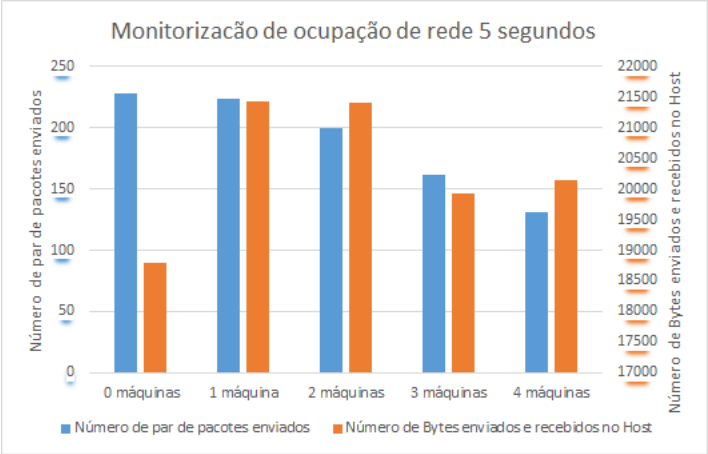
Este método foi aplicado para se verificar o número de pares de pacotes enviados (e recebidos) que o programa consegue efetuar, em situações de ocupação de rede distintas. Foi verificado com essa experiência que, apesar de o número de pacotes enviados aumentar conforme o período de tempo utilizado (como seria de esperar), existe uma relação inversa entre o número de pares de pacotes enviados/recebidos efetuados pelo programa e o número de máquinas que se encontram a utilizar a placa de rede. Ainda podemos concluir que esta relação é independente do tempo de execução do programa (como se pode verificar na imagem 3.7). Assim podemos garantir que, mesmo com um tempo reduzido de medição, é possível fazer uma análise fiável dos resultados.

Um conjunto completo com as imagens dos resultados pode ser consultado no anexo C. Os resultados obtidos com testes de monitorização de 5 segundos podem ser consultados na figura 3.8, uma vez que os resultados obtidos são fiáveis, e contém uma clara distinção entre os valores, sendo considerado o caso de execução mais favorável para o nosso sistema. Este caso de execução também reduz a necessidade de um tempo elevado de execução do programa.

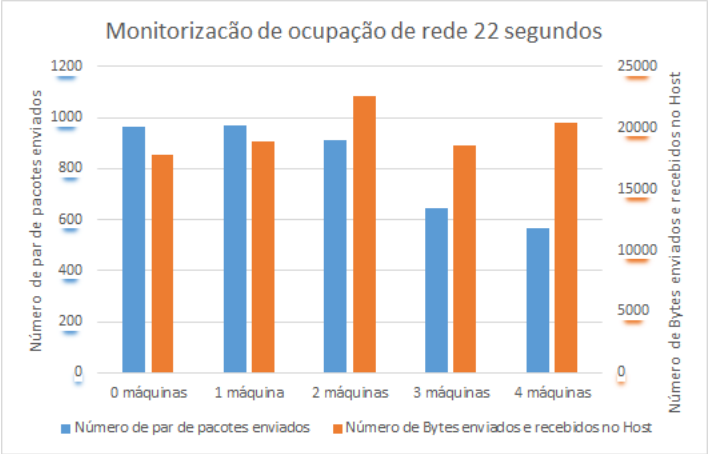
Na figura 3.8a podemos observar os valores obtidos diretamente dos testes efetuados, e na figura 3.8b encontram-se os valores das médias dos valores, de forma a facilitar a sua observação, assim como demonstrar a similaridade com a monitorização com outros valores de tempo. No anexo C as imagens da monitorização dos restantes tempos de execução podem ser observados, e encontram-se usando o mesmo tipo de ordenação.



(a) Testes de 2 segundos

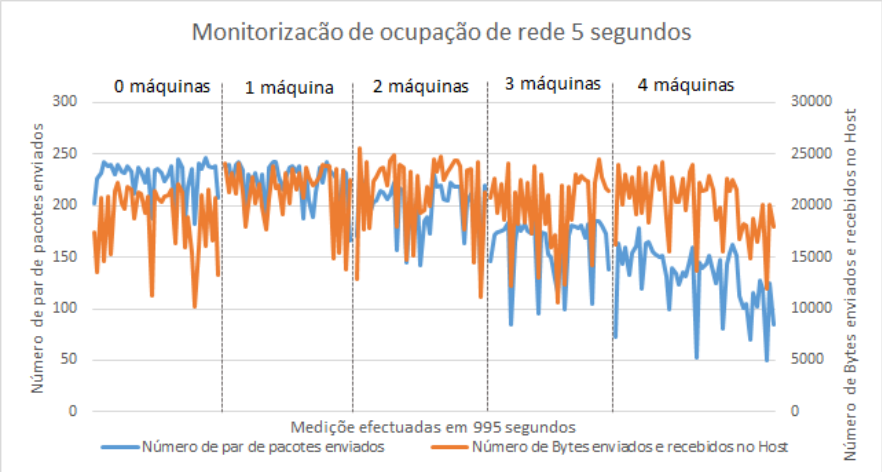


(b) Testes de 5 segundos

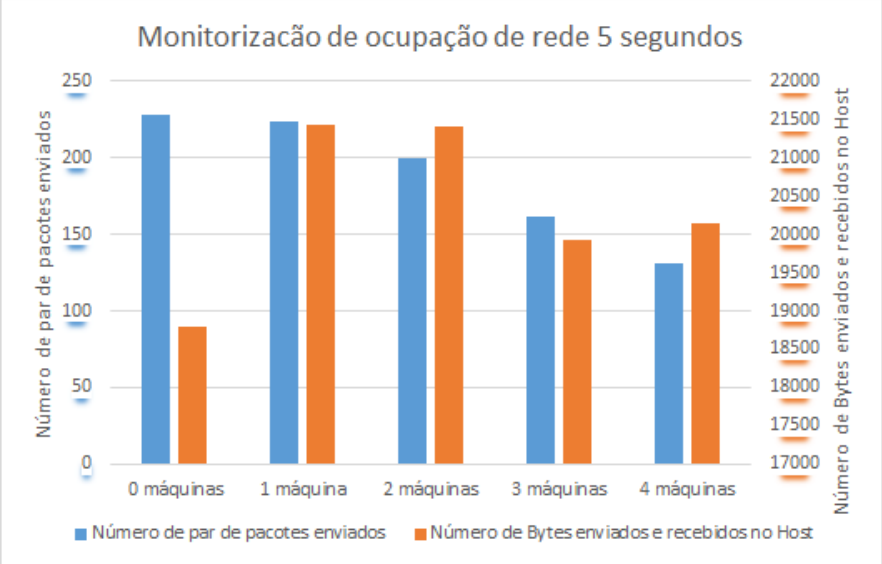


(c) Testes de 22 segundos

Figura 3.7: Comparação do número médio de parares de pacotes enviados *vs* Média do número de *Bytes* enviados e recebidos no *Host*, nos diversos tempos de medição de resultados.



(a) Número de par de pacotes enviados vs Número de Bytes enviados e recebidos no *Host*



(b) Média do número de par de pacotes enviados vs Média número de Bytes enviados e recebidos no *Host*

Figura 3.8: Resultados obtidos com a monitorização com intervalo fixo de 5 segundos, variando a ocupação da rede.

Com esta experiência é possível concluir que o número de pacotes enviados numa máquina virtual a correr num servidor com múltiplos clientes, é influenciado pelo estado de ocupação da rede dos restantes clientes, sendo portanto possível usar este métodos para se fazer a monitorização e previsão da indisponibilidade de recursos do servidor por uso excessivo da rede.

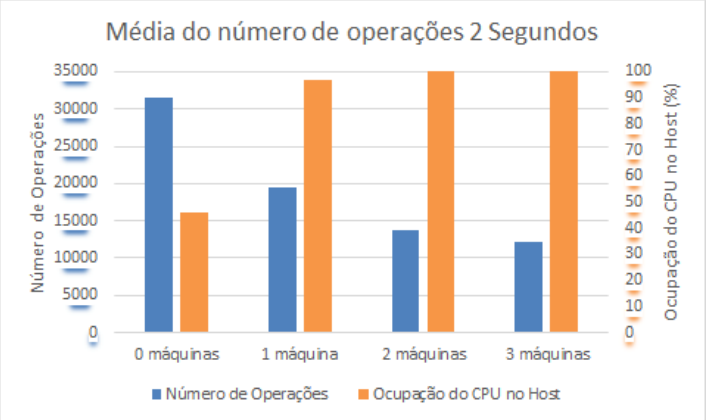
3.2.2 Monitorização da disponibilidade de recursos em OpenVZ

Para que os resultados obtidos sejam mais fiáveis, foram realizados os mesmos testes de desempenho num sistema a correr *OpenVZ*. O objetivo destes testes é a verificação da influência que o sistema de virtualização desempenha nos valores dos programas de monitorização.

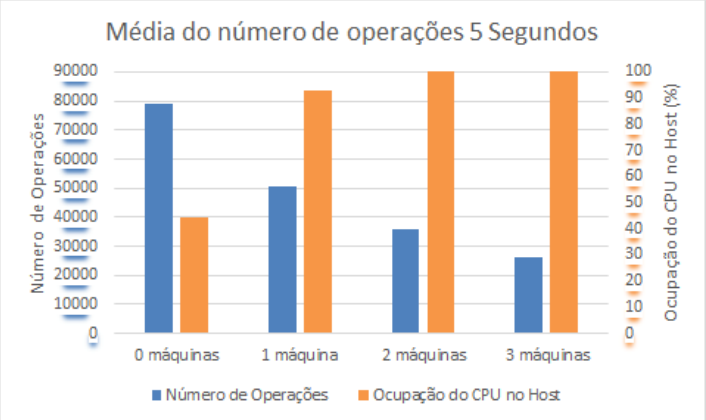
3.2.2.1 Monitorização do *CPU*

Utilizando o método descrito na secção 3.2.1.1 foram realizados testes de monitorização de *CPU* na mesma máquina a correr *OpenVZ*. Os testes realizados variavam unicamente no seu tempo de execução entre 2, 5 e 22 segundos, sendo a ocupação do *CPU* variada utilizando até três máquinas virtuais (como mencionado na secção 3.2.1.1).

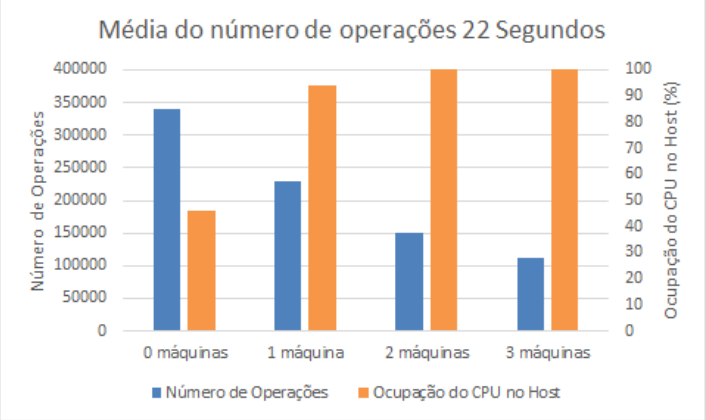
Como se verifica na figura 3.9, à semelhança dos resultados obtidos através do *KVM*, a ocupação do *CPU* no *Host* influencia o número de operações realizadas pela máquina virtual. Na figura 3.10 encontram-se os resultados para os testes de 5 segundos. Os restantes resultados encontram-se no anexo D.



(a) Testes de 2 segundos

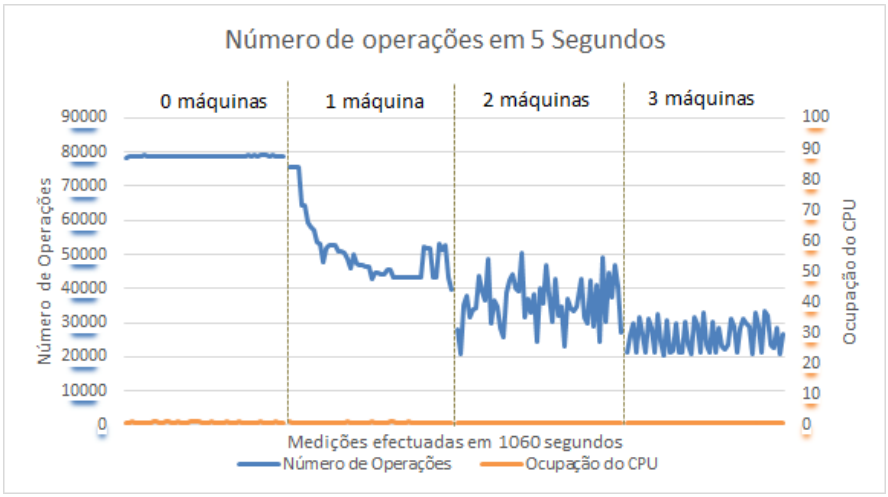


(b) Testes de 5 segundos

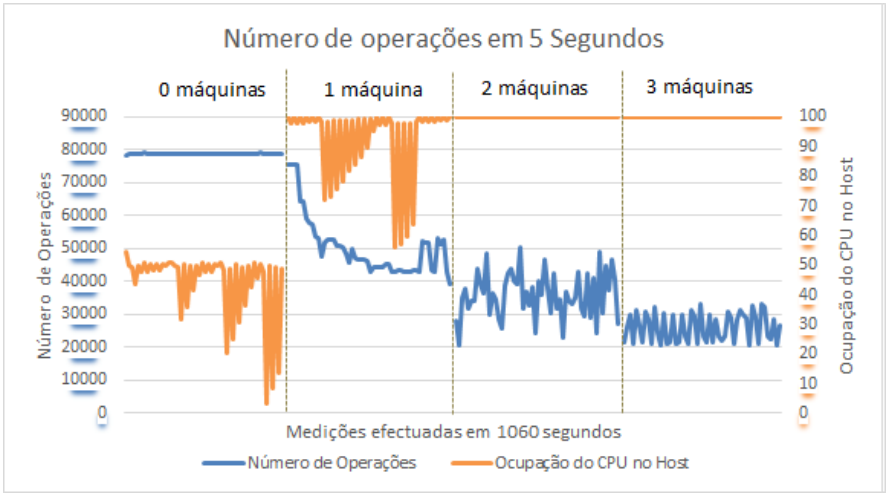


(c) Testes de 22 segundos

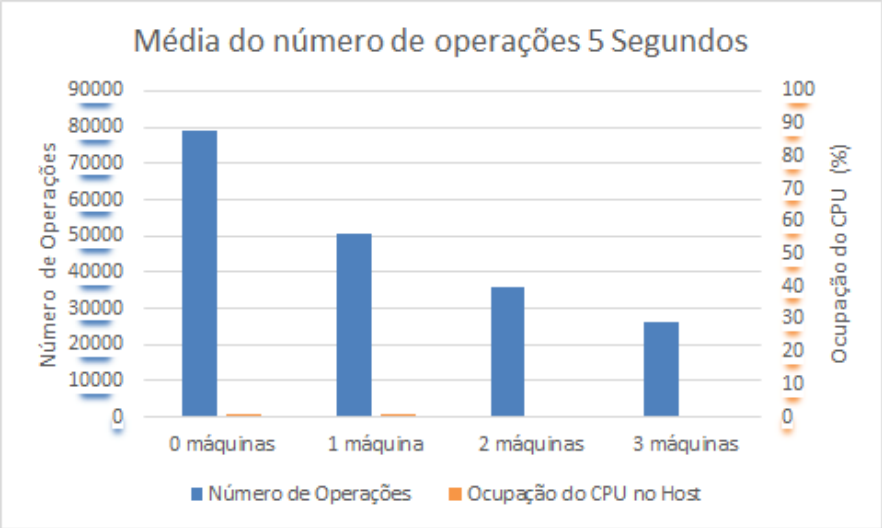
Figura 3.9: Comparação do número médio de Operações efetuadas *vs* Média da ocupação do *CPU* no *Host*, nos diversos tempos de medição de resultados.



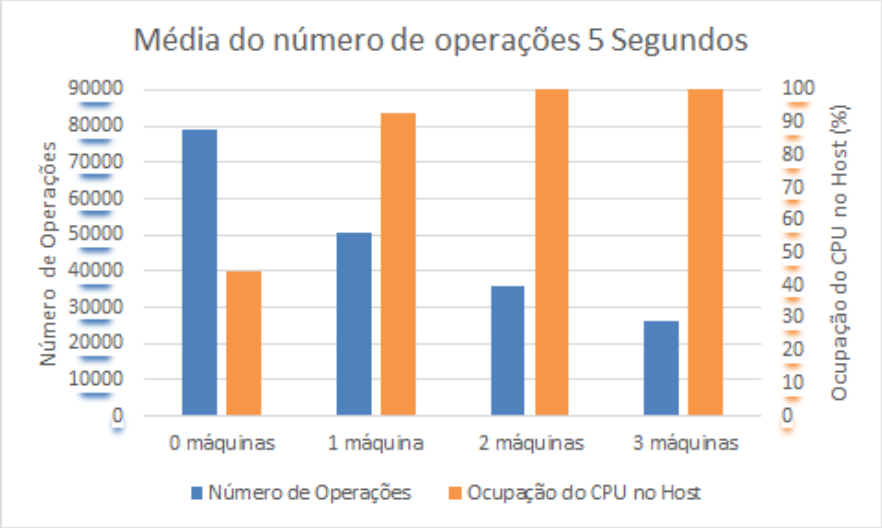
(a) Número de Operações efetuadas vs Medição de ocupação do *CPU* na máquina



(b) Número de Operações efetuadas vs Medição de ocupação do *CPU* no *Host*



(c) Número médio de Operações efetuadas vs Média da ocupação do *CPU* na máquina



(d) Número médio de Operações efetuadas vs Média da ocupação do *CPU* no *Host*

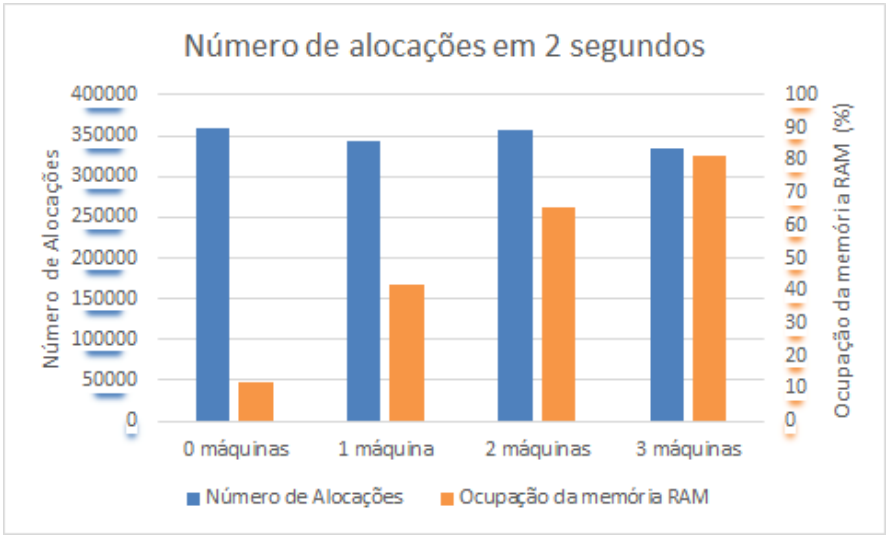
Figura 3.10: Resultados obtidos com a monitorização com intervalo fixo de 5 segundos, variando a ocupação do *CPU*.

3.2.2.2 Monitorização da memória *RAM*

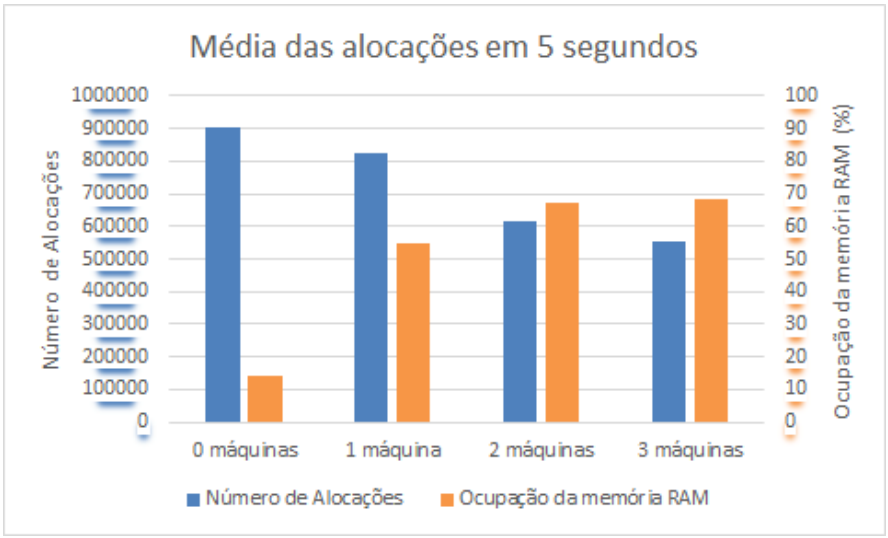
Utilizando o método descrito na secção 3.2.1.2 foram realizados testes de monitorização da memória *RAM* na mesma máquina a correr *OpenVZ*. Os testes realizados variavam unicamente no seu tempo de execução entre 2 e 5 segundos, sendo a ocupação da memória *RAM* variada utilizando até duas máquinas virtuais (como descrito na secção 3.2.1.2).

Pelo facto do *OpenVZ* conter um sistema de monitorização de recursos distinto do *KVM*, não foi possível realizar os testes com um tempo de execução de 22 segundos. Isto ocorreu porque a memória utilizada pela máquina virtual era demasiado elevada, e o próprio *OpenVZ* enviava um código de *SIGKILL* para terminar o processo em execução.

Como se pode observar na figura 3.11, à semelhança dos resultados obtidos através do *KVM*, a ocupação da memória *RAM* no *Host* influencia o número de alocações realizadas pela máquina virtual. Na figura 3.12 encontram-se os resultados para os testes de 5 segundos. Os restantes resultados encontram-se no anexo E.

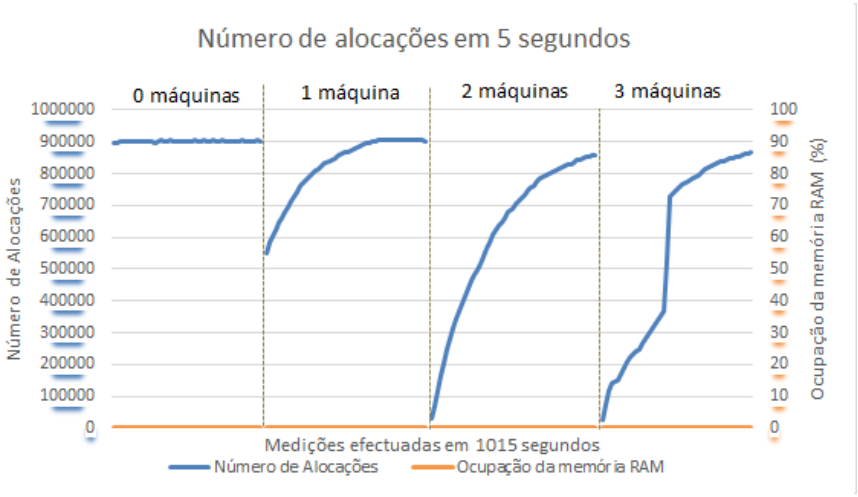


(a) Testes de 2 segundos

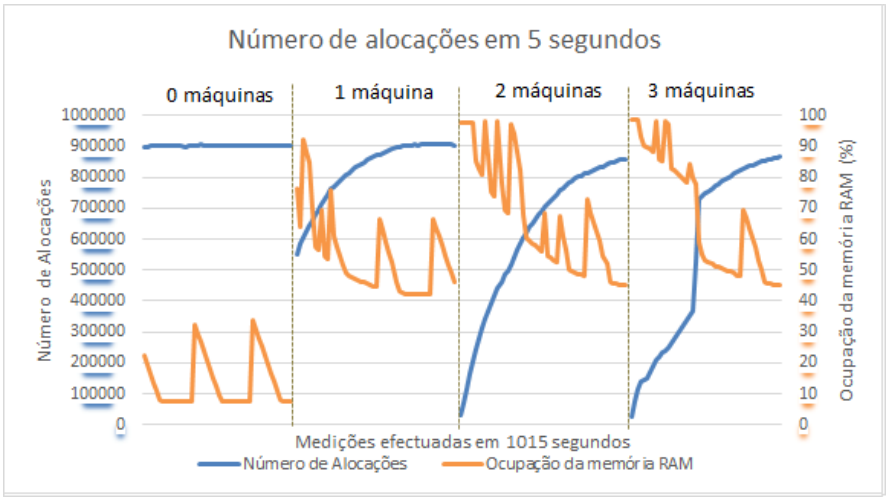


(b) Testes de 5 segundos

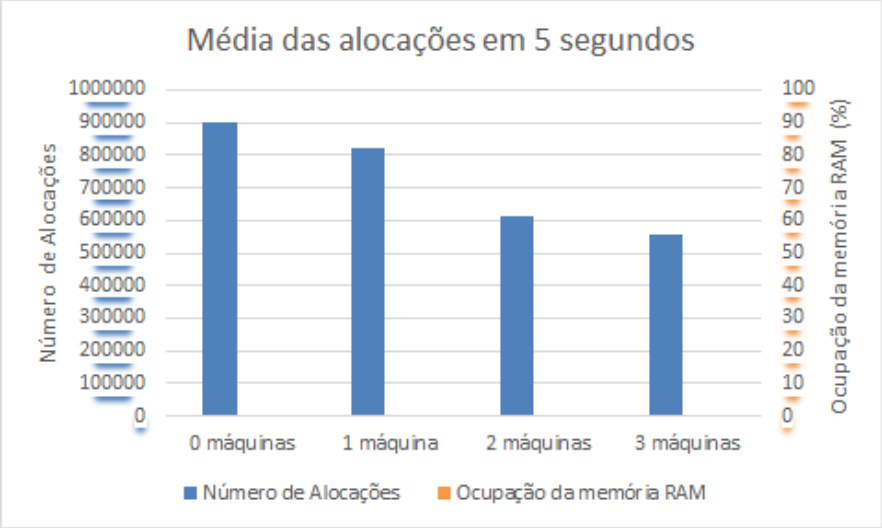
Figura 3.11: Comparação do número médio de alocações efetuadas *vs* Média da ocupação da memória *RAM* no *Host*, nos diversos tempos de medição de resultados.



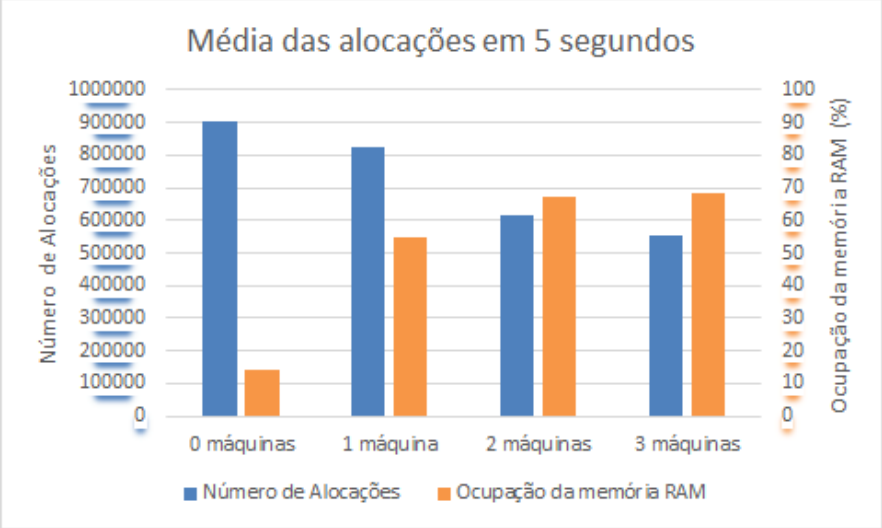
(a) Número de Operações efetuadas vs Medição de ocupação da memória *RAM* na máquina



(b) Número de Operações efetuadas vs Medição de ocupação da memória *RAM* no *Host*



(c) Número médio de Operações efetuadas vs Média da ocupação da memória *RAM* na máquina



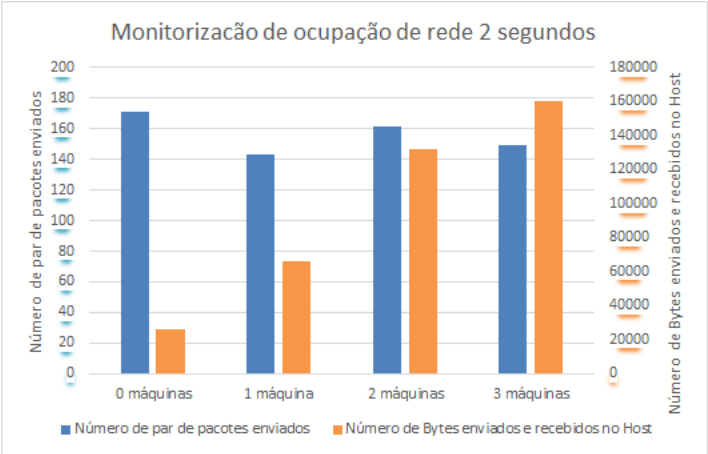
(d) Número médio de Operações efetuadas vs Média da ocupação da memória *RAM* no *Host*

Figura 3.12: Resultados obtidos com a monitorização com intervalo fixo de 5 segundos, variando a ocupação da memória *RAM*.

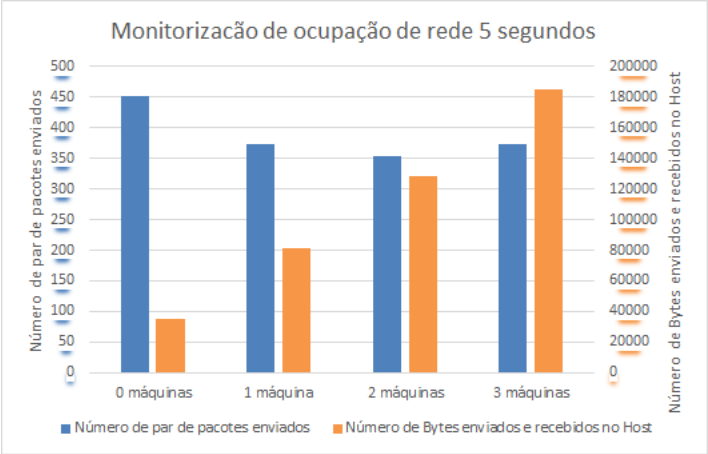
3.2.2.3 Monitorização de rede

Utilizando o método descrito na secção 3.2.1.3 foram realizados testes de monitorização da rede na mesma máquina a correr *OpenVZ*. Os testes realizados variavam unicamente no seu tempo de execução entre 2, 5 e 22 segundos. A ocupação da rede variada utilizando até três máquinas virtuais.

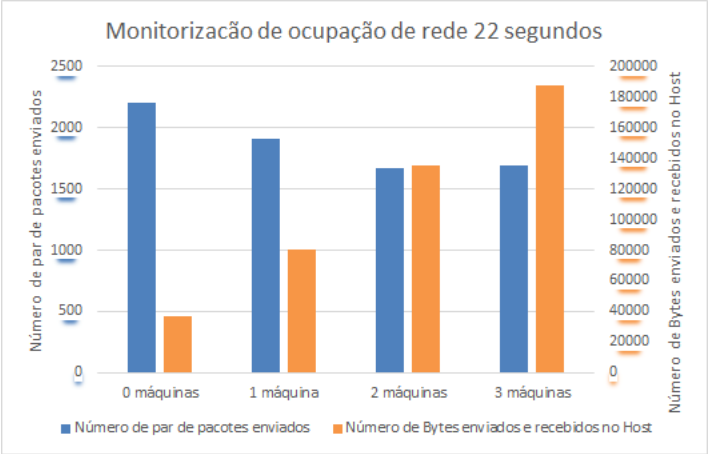
Como se observa na figura 3.13, à semelhança dos resultados obtidos através do *KVM*, a ocupação da rede no *Host* influencia o número de pacotes enviados pela máquina virtual. Na figura 3.14 encontram-se os resultados para os testes de 5 segundos. Os restantes resultados encontram-se no anexo F.



(a) Testes de 2 segundos

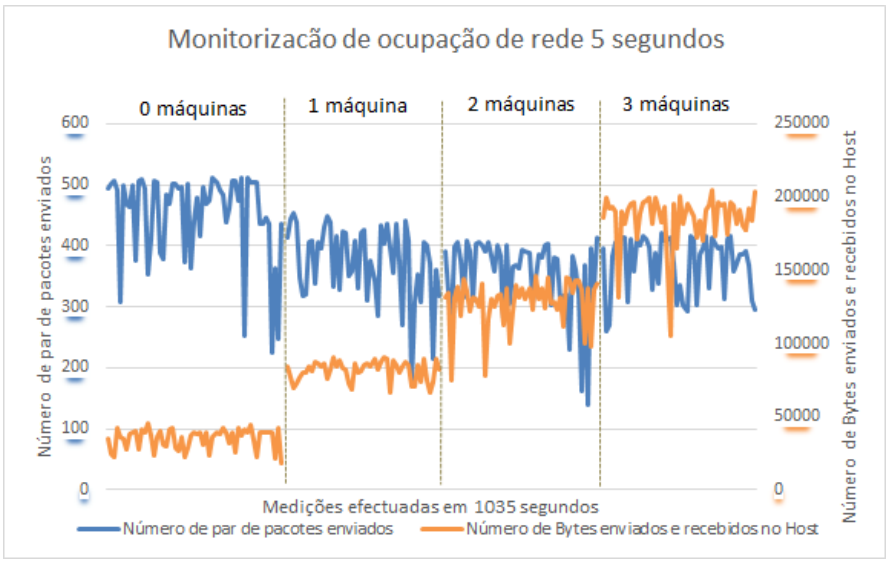


(b) Testes de 5 segundos

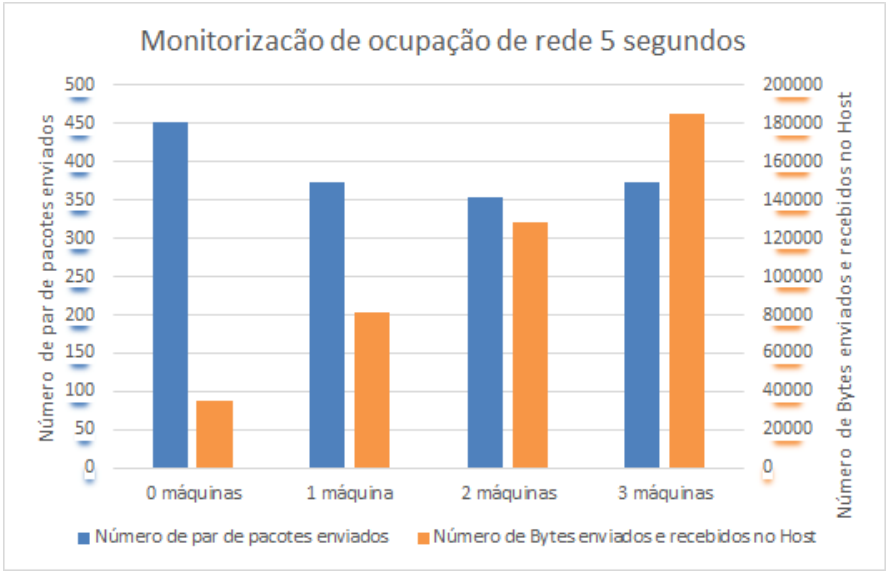


(c) Testes de 22 segundos

Figura 3.13: Comparação do número médio de pares de pacotes enviados *vs* Média da número de *Bytes* enviados e recebidos no *Host*, nos diversos tempos de medição de resultados.



(a) Número de par de pacotes enviados vs Número de Bytes enviados e recebidos no *Host*



(b) Média do número de par de pacotes enviados vs Média número de Bytes enviados e recebidos no *Host*

Figura 3.14: Resultados obtidos com a monitorização com intervalo fixo de 5 segundos, variando a ocupação da rede.

3.2.3 Comparação do *KVM* com *OpenVZ*

Com estes testes verificamos que, mesmo com as diferenças entre o *OpenVZ* e o *KVM*, é possível realizar os mesmos testes para se proceder a monitorização das principais componentes dos servidores. Deste modo, existe uma relação clara entre o estado de ocupação das componentes e o número de operações realizadas por estas. Estes valores fortalecem a hipótese de que é possível fazer a verificação do estado de ocupação de uma componente de um servidor através de uma máquina virtual a correr nesse servidor.

4 | Previsão da Indisponibilidade de Recursos

Na previsão da indisponibilidade de recursos, foram analisados os valores obtidos, sendo realizada uma tentativa de prever se os seguintes valores de monitorização se encontrarão abaixo dos valores esperados para o correto funcionamento do sistema. Desta forma, espera-se garantir que, em qualquer circunstância, o sistema esteja a ser executado sem qualquer problema e no pleno das suas capacidades, uma vez que se tenta resolver o problema mesmo antes deste acontecer. Como o objetivo de prever esses valores, são utilizados algoritmos de regressão, que permitem encontrar funções que melhor se ajustam aos pontos obtidos nas monitorizações anteriores.

Neste documento entende-se como indisponibilidade de recursos (falha) do servidor um evento que reduza, de forma significativa, o seu desempenho (p.e. uma elevada utilização do processador que cause aumento no tempo de execução de programas/instruções), e não um evento que cause o término de operações do servidor.

4.1 Regressão Polinomial

Os algoritmos de regressão polinomial são uma forma de regressão linear, onde a relação entre a variável independente x e a variável dependente y é modelada com uma equação polinomial de grau n.

O objetivo da análise da regressão é fazer um modelo dos valores esperados pela variável dependente y em relação à variável independente x. De uma forma geral, podemos especificar que uma equação polinomial de grau n é representada pela equação 4.1.

$$y = a_0x^0 + a_1x^1 + a_2x^2 + a_3x^3 + a_4x^4 + (...) + a_{n-1}x^{n-1} + a_nx^n \tag{4.1}$$

Assim, a regressão linear de grau n tem como objetivo encontrar os valores de a_0, a_1, a_2, a_3, a_4, ..., a_{n-1} e a_n, que tornem a equação 4.1 o mais próxima possível de um dado conjunto de pontos fornecido. Deste modo é possível fazer previsões sobre os valores de monitorização seguintes.

O caso em que o grau da variável independente x toma apenas um valor (de grau igual a um), denomina-se de regressão linear simples (descrito na secção 4.1.1). Quando toma mais do que um valor é denominado de regressão linear múltipla (dos quais será descrito o caso em que a equação é quadrática na secção 4.1.2).

Os valores de a_0 até a_n na regressão polinomial são obtidos através da resolução da equação entre matrizes *[x]* e *[a]*, demonstrada na equação 4.2, onde N é o número de pontos do conjunto para o qual se quer a equação polinomial que melhor se adeque, n é o grau da equação polinomial.[18–21]

$$\begin{bmatrix} \sum_{i=1}^{N} x_i^0 & \sum_{i=1}^{N} x_i^1 & \dots & \sum_{i=1}^{N} x_i^{n-1} & \sum_{i=1}^{N} x_i^n \\ \sum_{i=1}^{N} x_i^1 & \sum_{i=1}^{N} x_i^2 & \dots & \sum_{i=1}^{N} x_i^n & \sum_{i=1}^{N} x_i^{n+1} \\ \sum_{i=1}^{N} x_i^2 & \sum_{i=1}^{N} x_i^3 & \dots & \sum_{i=1}^{N} x_i^{n+1} & \sum_{i=1}^{N} x_i^{n+2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \sum_{i=1}^{N} x_i^{n-1} & \sum_{i=1}^{N} x_i^n & \dots & \sum_{i=1}^{N} x_i^{2n-2} & \sum_{i=1}^{N} x_i^{2n-1} \\ \sum_{i=1}^{N} x_i^n & \sum_{i=1}^{N} x_i^{n+1} & \dots & \sum_{i=1}^{N} x_i^{2n-1} & \sum_{i=1}^{N} x_i^{2n} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} x_i^0 y_i \\ \sum_{i=1}^{N} x_i^1 y_i \\ \sum_{i=1}^{N} x_i^2 y_i \\ \vdots \\ \sum_{i=1}^{N} x_i^{n-1} y_i \\ \sum_{i=1}^{N} x_i^n y_i \end{bmatrix} \tag{4.2}$$

Na equação 4.2 o primeiro valor $\sum_{i=1}^{N} x_i^0$ pode ser substituído por N, uma vez que x_i^0 é igual a *1*, para qualquer valor de i, da mesma forma que o somatório $\sum_{i=1}^{N} x_i^0 y_i$ pode ser substituído por $\sum_{i=1}^{N} y_i$.

Tendo em conta que o objetivo é obter os valores de a_0 até a_n a equação 4.2 pode ser simplificada de forma a obtermos a equação 4.3, que origina o sistema de equações 4.4.

$$\begin{bmatrix} a_0 \sum_{i=1}^{N} x_i^0 & a_1 \sum_{i=1}^{N} x_i^1 & \dots & a_{n-1} \sum_{i=1}^{N} x_i^{n-1} & a_n \sum_{i=1}^{N} x_i^n \\ a_0 \sum_{i=1}^{N} x_i^1 & a_1 \sum_{i=1}^{N} x_i^2 & \dots & a_{n-1} \sum_{i=1}^{N} x_i^n & a_n \sum_{i=1}^{N} x_i^{n+1} \\ a_0 \sum_{i=1}^{N} x_i^2 & a_1 \sum_{i=1}^{N} x_i^3 & \dots & a_{n-1} \sum_{i=1}^{N} x_i^{n+1} & a_n \sum_{i=1}^{N} x_i^{n+2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_0 \sum_{i=1}^{N} x_i^{n-1} & a_1 \sum_{i=1}^{N} x_i^n & \dots & a_{n-1} \sum_{i=1}^{N} x_i^{2n-2} & a_n \sum_{i=1}^{N} x_i^{2n-1} \\ a_0 \sum_{i=1}^{N} x_i^n & a_1 \sum_{i=1}^{N} x_i^{n+1} & \dots & a_{n-1} \sum_{i=1}^{N} x_i^{2n-1} & a_n \sum_{i=1}^{N} x_i^{2n} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} x_i^0 y_i \\ \sum_{i=1}^{N} x_i^1 y_i \\ \sum_{i=1}^{N} x_i^2 y_i \\ \vdots \\ \sum_{i=1}^{N} x_i^{n-1} y_i \\ \sum_{i=1}^{N} x_i^n y_i \end{bmatrix} \tag{4.3}$$

$$\begin{cases} a_0 \sum_{i=1}^{N} x_i^0 + a_1 \sum_{i=1}^{N} x_i^1 + \ldots + a_{n-1} \sum_{i=1}^{N} x_i^{n-1} + a_n \sum_{i=1}^{N} x_i^n = \sum_{i=1}^{N} x_i^0 y_i \\ a_0 \sum_{i=1}^{N} x_i^1 + a_1 \sum_{i=1}^{N} x_i^2 + \ldots + a_{n-1} \sum_{i=1}^{N} x_i^n + a_n \sum_{i=1}^{N} x_i^{n+1} = \sum_{i=1}^{N} x_i^1 y_i \\ a_0 \sum_{i=1}^{N} x_i^2 + a_1 \sum_{i=1}^{N} x_i^3 + \ldots + a_{n-1} \sum_{i=1}^{N} x_i^{n+1} + a_n \sum_{i=1}^{N} x_i^{n+2} = \sum_{i=1}^{N} x_i^2 y_i \\ \vdots \\ a_0 \sum_{i=1}^{N} x_i^{n-1} + a_1 \sum_{i=1}^{N} x_i^n + \ldots + a_{n-1} \sum_{i=1}^{N} x_i^{2n-2} + a_n \sum_{i=1}^{N} x_i^{2n-1} = \sum_{i=1}^{N} x_i^{n-1} y_i \\ a_0 \sum_{i=1}^{N} x_i^n + a_1 \sum_{i=1}^{N} x_i^{n+1} + \ldots + a_{n-1} \sum_{i=1}^{N} x_i^{2n-1} + a_n \sum_{i=1}^{N} x_i^{2n} = \sum_{i=1}^{N} x_i^n y_i \end{cases} \tag{4.4}$$

Como o sistema 4.4 contém *n+1* equações e *n+1* variáveis, este é um sistema possível de resolver.

4.1.1 Regressão Linear Simples

A regressão linear simples é uma forma de regressão polinomial de grau *1*, que tenta encontrar a função linear do género $y = ax + b$ que melhor se adeque a um dado conjunto de pontos.

A fórmula utilizada para o cálculo do valor de a encontra-se na equação 4.5, e a fórmula para o valor de b encontra-se na equação 4.6. Ambos os valores são derivados da fórmula da regressão polinomial onde a e b correspondem a a_1 e a_0 respetivamente.

$$a = \frac{\sum_{i=1}^{N} x_i y_i - \frac{1}{N}\sum_{i=1}^{N} x_i \sum_{j=1}^{N} y_j}{\sum_{i=1}^{N}(x_i^2) - \frac{1}{N}(\sum_{i=1}^{N} x_i)^2} \tag{4.5}$$

$$b = \frac{\sum_{i=1}^{N} y_i - a\sum_{j=1}^{N} x_j}{N} \tag{4.6}$$

Nas duas equações, 4.5 e 4.6, N é o número total de pontos para os quais queremos encontrar a expressão linear que melhor se adequa.[21–23]

4.1.2 Regressão Quadrática

A regressão Quadrática é uma forma de regressão polinomial de grau *2*, que tenta encontrar a função linear do género $y = ax^2+bx+c$ que melhor se adequa a um dado conjunto de pontos.

A fórmula utilizada para o cálculo do valor de a encontra-se na equação 4.7, a fórmula para o valor de b encontra-se na equação 4.8 e a fórmula para o valor de c encontra-se na equação 4.9. Ambos os valores são derivados da fórmula da regressão polinomial onde a, b e c correspondem a a_2, a_1 e a_0 respetivamente.

$$a = -\frac{-\sum_{i=1}^{N} x_i^2 y_i (\sum_{i=1}^{N} x_i)^2 + \sum_{i=1}^{N} x_i \sum_{i=1}^{N} x_i^2 \sum_{i=1}^{N} x_i y_i + \sum_{i=1}^{N} x_i \sum_{i=1}^{N} x_i^3 \sum_{i=1}^{N} y_i}{\sum_{i=1}^{N} x_i^4 (\sum_{i=1}^{N} x_i)^2 - 2\sum_{i=1}^{N} x_i \sum_{i=1}^{N} x_i^2 \sum_{i=1}^{N} x_i^3}$$

$$\frac{-\sum_{i=1}^{N} y_i (\sum_{i=1}^{N} x_i^2)^2 + N\sum_{i=1}^{N} x_i^2 \sum_{i=1}^{N} x_i^2 y_i - N\sum_{i=1}^{N} x_i^3 \sum_{i=1}^{N} x_i y_i}{+(\sum_{i=1}^{N} x_i^2)^3 - N\sum_{i=1}^{N} x_i^2 \sum_{i=1}^{N} x_i^4 + N(\sum_{i=1}^{N} x_i^3)^2} \tag{4.7}$$

$$b = -\frac{-a\sum_{i=1}^{N} x_i \sum_{i=1}^{N} x_i^2 + aN\sum_{i=1}^{N} x_i^3 + \sum_{i=1}^{N} x_i \sum_{i=1}^{N} y_i - N\sum_{i=1}^{N} x_i y_i}{N\sum_{i=1}^{N} x_i^2 - (\sum_{i=1}^{N} x_i)^2} \tag{4.8}$$

$$c = \frac{-a\sum_{i=1}^{N} x_i^2 - b\sum_{i=1}^{N} x_i + \sum_{i=1}^{N} y_i}{N} \tag{4.9}$$

Nas três equações, 4.7, 4.8 e 4.8, N é o número total de pontos para os quais queremos encontrar a expressão linear que melhor se adequa.[18, 21, 24]

4.2 Regressão Exponencial

A regressão exponencial é uma forma de regressão não-linear, que tenta encontrar a função linear do género $y = be^{ax}$ que melhor se adequa a um dado conjunto de pontos.

A fórmula utilizada para o cálculo do valor de a encontra-se na equação 4.10, e a fórmula para o valor de b encontra-se na equação 4.11.

$$a = \frac{N\sum_{i=1}^{N} x_i ln(y_i) - \sum_{i=1}^{N} x_i \sum_{i=1}^{N} ln(y_i)}{N\sum_{i=1}^{N} x_i^2 - (\sum_{i=1}^{N} x_i)^2} \tag{4.10}$$

$$b = e^{\frac{\sum_{i=1}^{N} ln(y_i)\sum_{i=1}^{N} x_i^2 - \sum_{i=1}^{N} x_i \sum_{i=1}^{N} x_i ln(y_i)}{N\sum_{i=1}^{N} x_i^2 - (\sum_{i=1}^{N} x_i)^2}} \tag{4.11}$$

Nas duas equações, 4.10 e 4.11, N é o número total de pontos para os quais queremos encontrar a expressão linear que melhor se adequa.[25, 26]

4.3 Coeficiente de Determinação

O coeficiente de determinação, geralmente denominado por R^2, é um número que indica a proximidade da variável dependente que é prevista da variável dependente que é fornecida. O R^2 varia entre 0 e 1, indicando o quanto o modelo se consegue ajustar aos valores observados. Para um maior valore de R^2, melhor o ajuste da precisão dos dados fornecidos.[18]

Este valor é usado para determinar quais dos métodos de regressão mencionados anteriormente (4.1.1, 4.1.2 e 4.2), melhor se adequa aos dados fornecidos.

Para obtermos o valor de R^2 temos que utilizar as equações 4.12, 4.13, 4.14, 4.15, 4.16 e 4.17, onde N é o número total de valores fornecidos, y_i é o valor fornecido de y no índice i e $\hat{y}_i$ é o valor previsto pela equação de regressão de y no índice i.

Nestas equações, SQ_{tot} é a soma total dos quadrados, ou seja, a soma dos quadrados das diferenças entre a média e cada valor observado. SQ_{exp} é a soma dos quadrados explicada, e indica-nos a diferença entre a média das observações e o valor estimado para cada observação, quanto menor for a diferença, maior poder explicativo detém a equação de regressão. SQ_{res} é

a soma dos quadrados dos resíduos, que calcula a parte que não é explicada pela equação de regressão.

$$\bar{y} = \frac{\sum_{i=1}^{N} y_i}{N} \tag{4.12}$$

$$SQ_{tot} = \sum_{i=1}^{N} (y_i - \bar{y})^2 \tag{4.13}$$

$$SQ_{exp} = \sum_{i=1}^{N} (\hat{y_i} - \bar{y})^2 \tag{4.14}$$

$$SQ_{res} = \sum_{i=1}^{N} (\hat{y_i} - y_i)^2 \tag{4.15}$$

$$SQ_{tot} = SQ_{exp} + SQ_{res} \tag{4.16}$$

$$R^2 = \frac{SQ_{exp}}{SQ_{tot}} = 1 - \frac{SQ_{res}}{SQ_{tot}} \tag{4.17}$$

No caso da regressão exponencial (4.2), uma vez que se trata de uma equação não linear, este método não pode ser aplicado diretamente. No entanto, se à fórmula da equação exponencial ($y = be^{ax}$) aplicarmos a operação logarítmica, obtemos a expressão $ln(y) = ln(b) + ax$, tornando-se assim uma expressão linear. Deste modo, aplicando a operação logarítmica em todos os valores de Y, é possível determinar o coeficiente de determinação da regressão exponencial.

4.4 Previsão nos valores obtidos

Uma vez que, nos casos que iremos considerar, para a previsão da indisponibilidade de recursos, apenas temos uma única dimensão de valores (valores de processador, memória RAM ou rede), estes valores são utilizados como valores do eixo do y, enquanto os valores do eixo do x são incrementados sequencialmente (deste *1* até N). O valor de monitorização mais antigo terá um valor de x menor do que um valor de monitorização mais recente. Por exemplo, considere-se os valores obtidos numa monitorização de ocupação do processador numa máquina virtual a correr em KVM, ao mesmo tempo que outras máquinas a correr no mesmo sistema realizavam operações de ocupação de processador aleatoriamente, que se encontram na tabela 4.1. Os valores estão ordenados horizontalmente por momento da monitorização, ou seja, o primeiro valor encontra-se no canto superior esquerdo, e o último valor encontra-se no canto inferior direito.

58404	81051	80574	80627	81546	81321	81366	81595	44556	25833
45780	43454	51647	40901	47568	42555	46851	17433	31452	29114
27235	32113	40209	37187	33832	29914	32268	33627	70152	51071
51376	58588	49141	48747	49925	75145	73549	48535	50911	46931
49920	74727	76320	74208	76348	74887	74650	73918	74832	76470

Tabela 4.1: Valores obtidos na monitorização da ocupação do *CPU* de uma máquina virtual a correr em *KVM*, ordenados horizontalmente por momento da monitorização.

Para que seja possível aplicar as fórmulas de regressão, colocam-se estes valores no eixo do y e é incrementado o valor do eixo do x utilizando os valores temporais de quando as medições foram efetuadas. Assim, o valor *58404* fica representado no gráfico como o ponto *(1, 58404)*, o valor 81051 como o ponto *(2, 81051)*, ..., até ao valor 76470 que fica representado como o ponto *(50, 76470)*. Estes pontos encontram-se representados na figura 4.1. Em seguida, a estes pontos são aplicadas as fórmulas de regressão de forma a obtermos a expressão da equação que melhor se adequa aos pontos fornecidos.

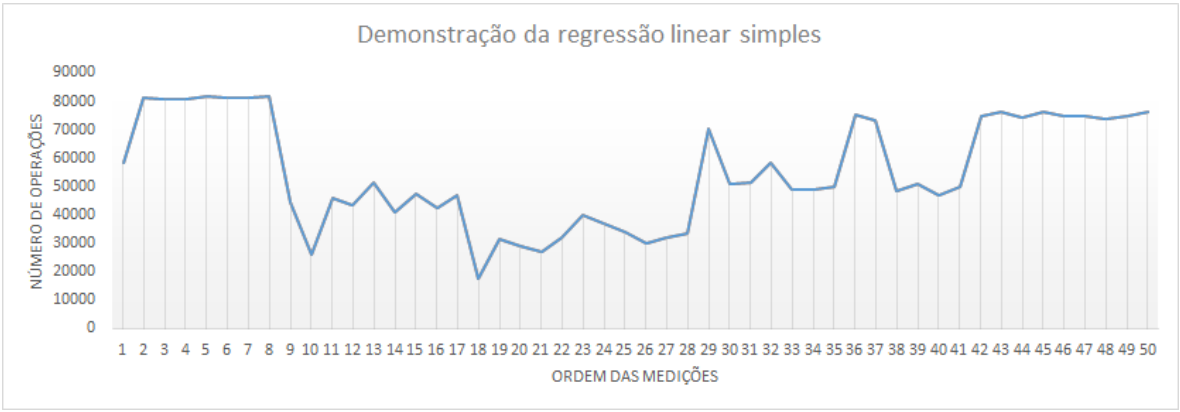


Figura 4.1: Representação dos valores da tabela 4.1 num gráfico cartesiano.

Nos exemplos seguintes aplicar-se-á apenas a regressão linear simples, uma vez que esta é a fórmula mais intuitiva e simples de demonstrar a previsão de valores. As restantes fórmulas de regressão seguem uma lógica semelhante utilizando as fórmulas descritas anteriormente. No programa de monitorização serão aplicados os três modelos de regressão e será utilizado aquele que tiver um valor de coeficiente de determinação maior (demonstrado na secção 4.4.1). Na figura 4.2 encontra-se demonstrada a reta da regressão linear aplicada aos pontos da tabela 4.1, assim como a equação obtida através deste método.

Esta equação oferece-nos uma forma simples de previsão de tendência dos valores que se seguirão nas próximas monitorizações, podendo ser aplicada para prever se os valores seguintes se encontrarão abaixo de um certo limite. Por exemplo, considerando que nos encontramos no momento em que o quadragésimo quinto valor foi medido, obteríamos o gráfico demonstrado na figura 4.3, assim como respetiva equação (e linha de tendência).

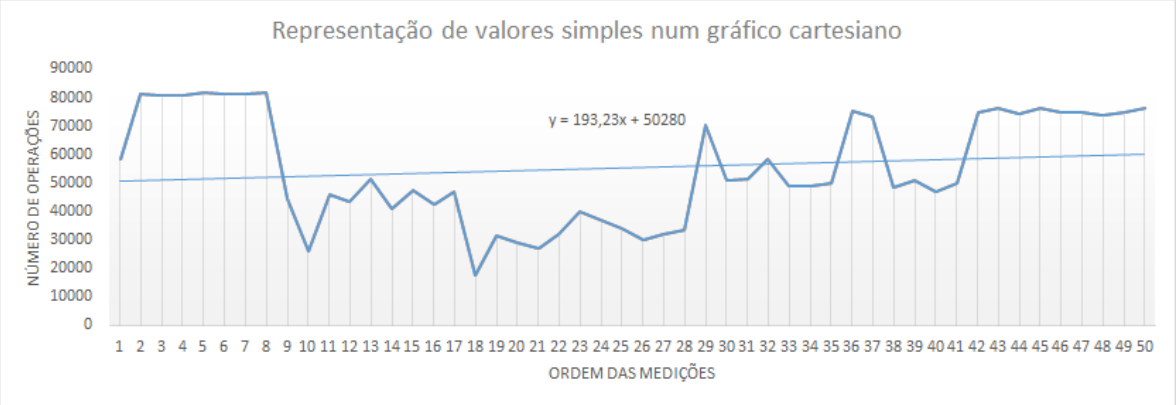


Figura 4.2: Demonstração da regressão linear simples.

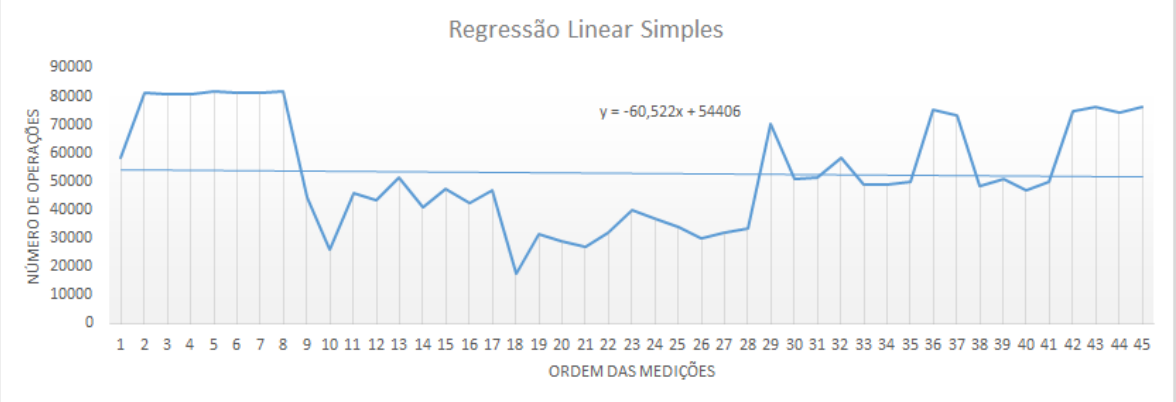
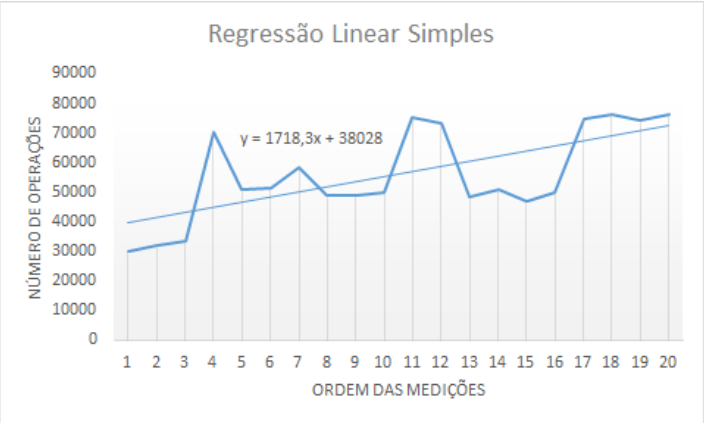
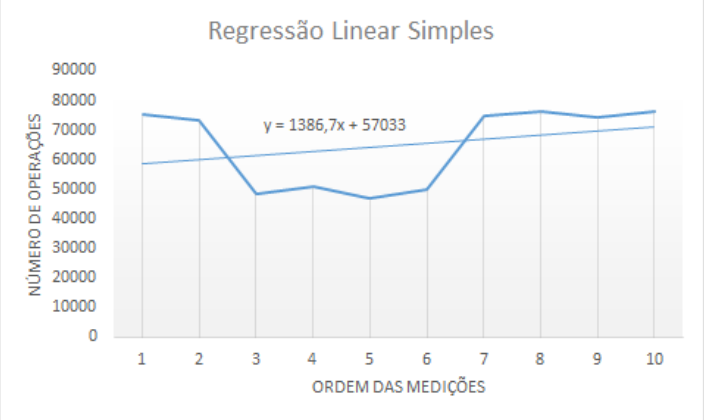


Figura 4.3: Regressão linear simples ao quadragésimo quinto valor.

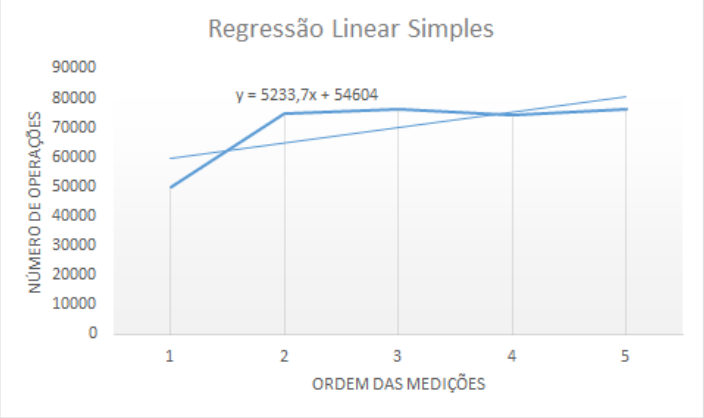
De forma a tentar prever os próximos valores (o número de medições seguintes que se pretende prever não deve ser muito elevado, uma vez que o sistema é bastante variável) utiliza-se a equação obtida pela fórmula da regressão linear simples e calcula-se o valor de y para os próximos valores de x. Ou seja, utilizando o exemplo da figura 4.3 onde se obteve a equação y = *-60,522x + 54406*, a previsão do valor que se obteria em seguida seria calculado substituindo o valor de x na equação por *46* (uma vez que o último valor medido é o ponto com o valor de *45* no eixo do x). O valor obtido, arredondado à casa das unidades, seria de *51622* operações. No entanto, como podemos verificar o valor seguinte que foi obtido pela monitorização foi de *74887* operações. Esta grande diferença prende-se no facto de que uma vez que estamos a considerar a grande maioria dos valores medidos, a equação da reta de previsão acaba por se tornar uma linha centrada na média de valores obtidos. Isto deve-se ao facto de as medições da ocupação do processador serem bastante variáveis e não lineares. Para evitar este problema, em vez de se usarem todos os valores existentes utilizam-se apenas um número limitado de valores anteriores. Na figura 4.4 encontram-se três gráficos que representam as últimas vinte, dez e cinco medições efetuadas no momento precedente à quadragésima quinta mediação.



(a) Representação dos últimos 20 resultados.



(b) Representação dos últimos 10 resultados.



(c) Representação dos últimos 5 resultados.

Figura 4.4: Representações dos últimos resultados obtidos com a monitorização da ocupação do processador no momento da quadragésima quinta medição.

Realizando as mesmas operações de previsão que efetuamos para os valores da figura 4.3 para cada um dos casos representados nas figuras 4.4a, 4.4b e 4.4c, obtemos os resultados *74112*, *72286* e *86006* respetivamente. Em comparação com os resultados obtidos anteriormente, estes já se encontram bastante mais próximos do resultado real obtido, uma vez que têm apenas em conta as últimas variações do estado de ocupação do processador. Neste exemplo, o caso em que medimos os vinte resultados anteriores obteve o valor mais aproximado. Este valor não significa que esse caso seja sempre o melhor. Para valores de previsão mais afastados do caso atual ou para alturas de medição diferentes este pode não ser o mais adequado. Por exemplo, nas mesmas condições queríamos prever o quadragésimo oitavo resultado (ou seja três medições depois do último resultado medido) com um valor de *73918* operações, cada um dos casos da figura 4.4 iriam prever valores de *77548*, *75060* e *96473* respetivamente. Neste caso a equação obtida utilizando as últimas dez medições proporcionava o melhor resultado. Da mesma forma, se quiséssemos prever o trigésimo resultado (com valor igual a *51376* operações), logo em seguida a se ter realizado a vigésima nona medição, utilizando as últimas vinte, dez e cinco medições, obteríamos os valores *37568*, *49252* e *62864* respetivamente, sendo o caso das últimas cinco medições o mais favorável.

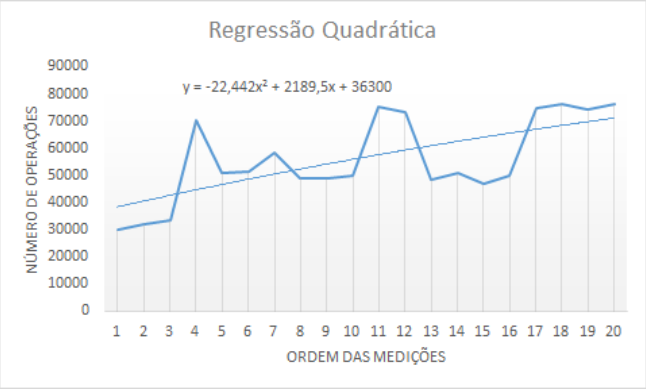
Com estes exemplos, pretende-se realçar que nunca se poderá fazer uma previsão que seja 100% eficaz. Também se verifica que o mesmo método de previsão pode não ser o mais favorável para todos os casos. Neste sistema, uma vez que o objetivo da previsão é verificar se o estado de ocupação do processador se vai encontrar acima ou abaixo de um certo limiar, utilizam-se as três variações de previsão dos resultados (com as vinte, dez e cinco medições anteriores). Assume-se como verdadeiro o caso que tiver duas ou mais previsões (caso 2 dos casos sejam acima e 1 seja abaixo o resultado final é acima).

4.4.1 Demonstração do coeficiente de determinação

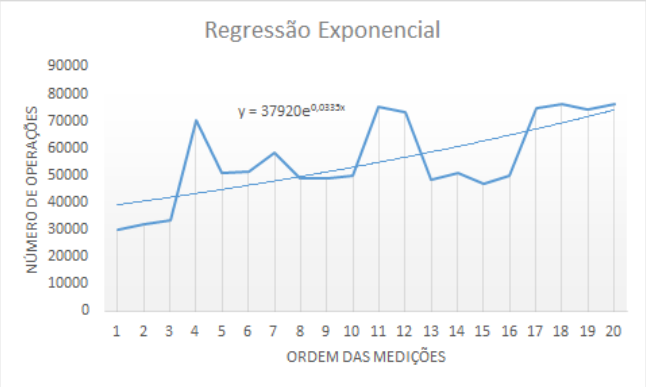
Na secção anterior foi utilizado unicamente a regressão linear simples por se tratar da fórmula de regressão mais intuitiva e simples de se demonstrar. Mas esta nem sempre é a fórmula que melhor se aplica aos pontos fornecidos.

Se tomarmos como exemplo os valores da figura 4.4 e aplicarmos o método de regressão quadrática e regressão exponencial, obtemos os resultados da figura 4.5.

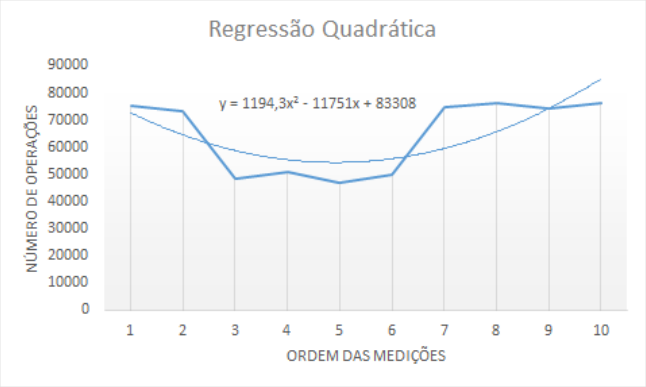
Ao aplicarmos a fórmula de cálculo do coeficiente de determinação para cada um dos casos, obtemos os valores da tabela 4.2. Utilizando esses valores chegamos a conclusão que, para o caso das últimas vinte medições, a regressão exponencial é aquela que melhor se ajusta aos pontos. Para os casos das últimas dez e das últimas cinco medições, a regressão quadrática é a que melhor se adapta, uma vez que possui um maior coeficiente de determinação.



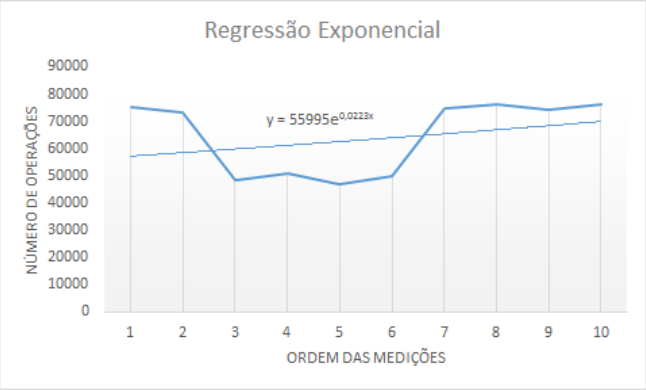
(a) Representação dos últimos 20 resultados.



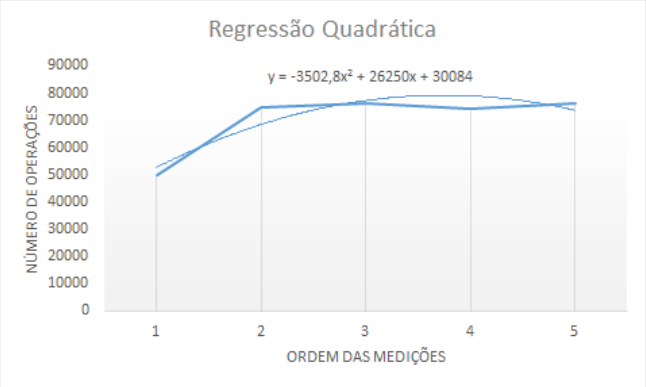
(b) Representação dos últimos 20 resultados.



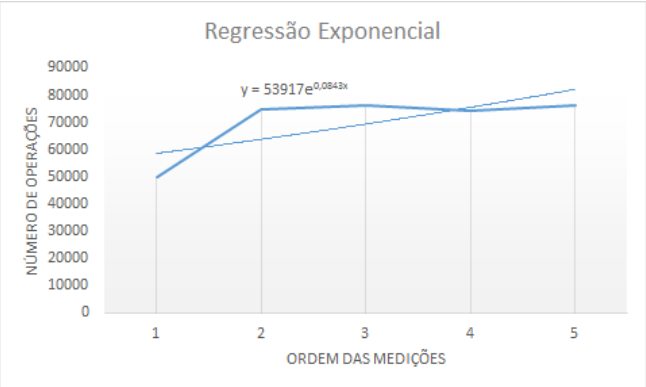
(c) Representação dos últimos 10 resultados.



(d) Representação dos últimos 10 resultados.



(e) Representação dos últimos 5 resultados.



(f) Representação dos últimos 5 resultados.

Figura 4.5: Resultados obtidos com a monitorização da ocupação do processador no momento da quadragésima quinta medição para diferentes fórmulas de regressão.

	Últimas 20 medições	**Últimas 10 medições**	**Últimas 5 medições**
Regressão linear simples	0,4346	0,0970	0,5237
Regressão quadrática	0,4365	0,5578	0,8521
Regressão exponencial	0,4572	0,0936	0,5198

Tabela 4.2: Valores do coeficiente de determinação.

5 | Descrição do Sistema de Monitorização da Disponibilidade e Previsão da Indisponibilidade de Recursos

O sistema desenvolvido divide-se em duas componentes principais:

1. Programa de monitorização
2. Servidor *Django*

A 1ª componente é um programa escrito em *Python*, cujo objetivo é fazer a monitorização dos servidores dos clientes (*Probes*), tal como descrito na secção 5.2.

A 2ª componente é um servidor *Django*, que foi desenvolvido pelo aluno José Luís da Silva Rosa no âmbito da sua dissertação de mestrado *Customer-Side Detection of BGP Routing Attacks*[1], ao qual foram adicionadas algumas aplicações de forma a permitir fazer o tratamento dos resultados obtidos pelo programa de monitorização, fazendo a previsão dos valores seguintes, tal como descrito na secção 5.4.2. Foi providenciada alguma assistência no desenvolvimento da interface *web*, Na criação do menu inicial e das páginas que principalmente se adaptam, unicamente, ao contexto desta dissertação.

A componente 2 quando recebe os valores enviados pelo programa de monitorização, guarda-os e processa-os de forma a verificar se o servidor cliente se encontra operacional ou não. Em caso negativo, são feitas verificações para se analisar a possibilidade de se moverem algumas das funcionalidades desse servidor para um servidor vizinho, que seja capaz de suportar as funcionalidades requeridas.

5.1 Tecnologias Utilizadas

As tecnologias utilizadas foram unicamente a linguagem *Python* e um servidor *Django*, de forma a reduzir a carga que o próprio programa de monitorização possa ter no servidor onde

irá ser colocado a correr.

5.1.1 Python

A linguagem de programação utilizada foi a linguagem *Python*, pois trata-se de uma linguagem poderosa, rápida, capaz de correr em qualquer sistema e é *Open-Source*.[27]

o *Python* é uma linguagem de programação orientada a objetos poderosa e limpa. Algumas das suas principais características são:

- Utiliza uma sintaxe simples e elegante, tornando o código fácil de escrever e ler.
- É uma linguagem fácil de utilizar, o que faz com que seja simples colocar o programa a funcionar. Isto faz com que o *Python* seja ideal para desenvolver protótipos e outras tarefas de programação criadas com um propósito único, sem comprometer a sua manutenção.
- Contém uma grande biblioteca *standard* capaz de suportar muitas tarefas comuns de programação, tais como conexão a servidores *Web*, pesquisa de texto com expressões regulares, leitura e modificação de ficheiros, entre outros.
- O modo interativo do *Python* faz com que seja simples o teste de pequenos excertos de código. Existe também um ambiente de desenvolvimento agrupado chamado *IDLE*.
- É fácil de estender ao se estender módulos implementados em linguagens compiladas tais com *C* ou *C++*.
- Pode ser embutido noutras aplicações de forma a fornecer uma *interface* programável.
- Corre em diversos computadores e sistemas operativos.
- É livre em dois sentidos. Não tem custos associados ao *download* e uso da linguagem, ou á sua inclusão noutras aplicações. Pode ser modificado livremente e redistribuído, pois apesar de ser *copyrighted* está disponível de forma *Open-Source*.

Estas características tornam o *Python* uma linguagem fácil de utilizar sem necessitar de elevados recursos do sistema.[28]

O uso de bibliotecas externas foi sempre evitado, de forma a que estas não tenham que ser instaladas em todos os sistemas que pretendam correr este programa, sendo apenas necessário que exista o *Python* nativo instalado nos servidores dos clientes.

5.1.2 Django

O *Django* é uma *Python Web framework* de alto nível que incentiva um desenvolvimento rápido e um design limpo e pragmático. Construído por programadores experientes, trata por

si só muitos dos inconvenientes do desenvolvimento *web*, para que se possa focar em escrever a aplicação sem necessidade de reinventar a roda. É grátis e *Open-Source*.

É extremamente rápido, pois foi desenhado para ajudar os programadores a desenvolver aplicações desde o conceito, até à conclusão, o mais rápido possível.

É muito seguro e ajuda os programadores a evitar erros de segurança comuns, tais como *SQL injection*, *cross-site scripting*, *cross-site request forgery* e *clickjacking*. O seu sistema de autenticação dos utilizadores proporciona uma forma segura de gerir as suas contas e respetivas *passwords*.

Tem a habilidade de ser escalável de forma rápida e flexível, de forma a responder a tráfego intenso.

Inclui dezenas de extras que podem ser utilizados para tratar de certas tarefas comuns no desenvolvimento *web*. Está dotado, logo de inicio, de tarefas tais como autenticação de utilizadores, administração de conteúdos, mapeamento do *site*, *feeds RSS*, e muitas outras tarefas.

É também muito versátil, sendo utilizado por diversas empresas, organizações e governos para diversos tipos de sistemas, incluindo manutenção, redes sociais, plataformas de computação cientifica, entre muitos outros. [29]

5.2 Arquitetura do Sistema

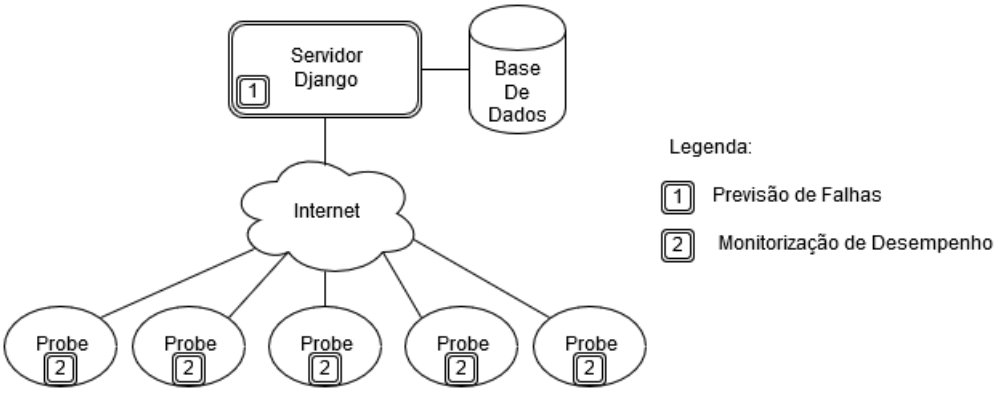


Figura 5.1: Arquitetura do sistema.

O sistema divide-se em três componentes principais:

1. Servidor *Django* contendo o algoritmo de previsão da indisponibilidade de recursos
2. Base de dados
3. *Probes* dos clientes contendo o sistema de monitorização da disponibilidade de recursos

A 1ª componente é a grande base do sistema. É responsável pelo envio do código de monitorização. Além disso, recebe e processa os resultados obtidos por cada uma das *probes* e efetua as operações necessárias para que, quando se verifique alguma anomalia, se possam realizar as respetivas operações de alteração dos servidores.

A 2ª componente é responsável pelo armazenamento dos dados relativos a cada uma das *probes* como por exemplo, os valores obtidos por cada monitorização, o estado da *probe* (ativo ou inativo), entre outros que serão descritos na secção 5.3.

A 3ª componente são os servidores dos clientes. O servidor *Django* envia o código de monitorização e inicia a sua execução. Posteriormente, o código de monitorização é efetuado e os respetivos resultados enviados de volta para o servidor.

5.3 Modelo de Dados

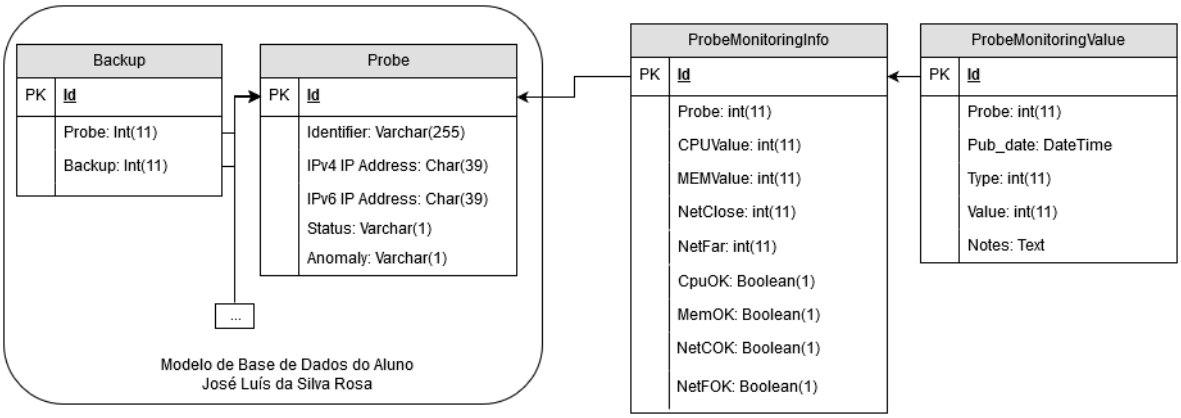


Figura 5.2: Modelo de dados do sistema.

O modelo de dados utilizado por este sistema integra-se de forma direta com o modelo de dados desenvolvido pelo aluno José Luís da Silva Rosa no âmbito da sua dissertação de mestrado *Customer-Side Detection of BGP Routing Attacks*[1]. Na figura 5.2 os modelos de dados da dissertação do José Rosa encontram-se no interior do quadrado, e os modelos desenvolvidos no âmbito desta dissertação encontra-se no exterior.

Os modelos de dados baseiam-se em dois principais conjuntos de dados:

1. ProbeMonitoringInfo
2. ProbeMonitoringValue

No modelo 1 são guardados os valores gerais sobre a capacidade computacional de cada cliente. Estes valores contêm a seguinte informação:

ProbeMonitoringInfo:

- **Probe** - Referência a uma *probe* única;
- **CPUValue** - Valor correspondente ao maior número do valor de monitorização do processador, realizadas por este servidor;
- **MEMValue** - Valor correspondente ao maior número do valor de monitorização da memória *RAM*, realizadas por este servidor;
- **NetClose** - Valor correspondente ao maior número do valor de monitorização da rede, para uma máquina próxima geograficamente deste servidor;
- **NetFar** - Valor correspondente ao maior número do valor de monitorização da rede, para uma máquina longínqua geograficamente deste servidor;
- **CpuOK** - Valor que indica se o estado de ocupação do processador se encontra dentro dos valores aceitáveis para o bom funcionamento do servidor;
- **MemOK** - Valor que indica se o estado de ocupação da memória *RAM* se encontra dentro dos valores aceitáveis para o bom funcionamento do servidor;
- **NetCOK** - Valor que indica se o estado de ocupação da rede para máquinas próximas se encontra dentro dos valores aceitáveis para o bom funcionamento do servidor;
- **NetFOK** - Valor que indica se o estado de ocupação da rede para máquinas longínquas se encontra dentro dos valores aceitáveis para o bom funcionamento do servidor.

No modelo 2 são guardados todos os valores de monitorização obtidos de cada cliente. Cada grupo de dados contém a seguinte informação:

ProbeMonitoringValue:

- **Probe** - Referência a uma *probe* única;
- **Pub_date** - Data e hora da altura que o valor de monitorização foi recebido;
- **Type** - Valor que indica que tipo de informação foi recebida pelo servidor (0 - Monitorização do CPU, 1 - Monitorização da memória *RAM*, 2 - Monitorização da rede para máquinas próximas, 3 - Monitorização da rede para máquinas longínquas);
- **Value** - O valor de monitorização obtido;
- **Notes** - Campo que contém notas adicionais (como por exemplo, previsão da indisponibilidade de recursos do servidor).

Do trabalho realizado no âmbito da dissertação do aluno José Rosa[1] são utilizadas as duas tabelas que se encontram detalhadas. Devido a motivos de privacidade e confidencialidade, as restantes tabelas não se encontram descritas uma vez que não são utilizadas no contexto desta dissertação. As duas tabelas utilizadas são:

Probe:

- **Identifier** - Identificador único de cada *probe*;
- **IPv4 IP Address** - Endereço *IPv4* da *probe*;
- **IPv6 IP Address** - Endereço *IPv6* da *probe*;
- **Status** - Valor que indica o estado da *probe*. Caso seja 0 então é uma probe de *backup* se for 1 a *probe* encontra-se ativa;
- **Anormaly** - Valor que indica se esta *probe* alguma vez sofreu de uma anormalia.

Backup:

- **Probe** - Referência para a *probe* principal;
- **Backup** - Referência para a *probe* de *backup*.

5.4 Programa desenvolvido

Como já foi referido anteriormente, o sistema desenvolvido divide-se em duas competentes principais: o programa de monitorização da disponibilidade de recursos e o programa de previsão da indisponibilidade de recursos. A comunicação entre as componentes é realizada através do envio de pedidos *HTTP* do programa de monitorização para o programa de previsão, onde são enviadas as informações necessárias.

Estes programas e respetivas componentes serão descritos nas secções seguintes.

5.4.1 Programa de monitorização da disponibilidade de recursos

Depois de realizados todos os testes referidos no capitulo 3, foi criado um programa cujo propósito era de realizar este tipo de monitorização de forma contínua nos servidores que se pretende monitorizar. A esse programa foi dado o nome de *VPSMonitorization*. Este programa corre de forma contínua, executando as monitorizações de processador, memória *RAM* e de rede por esta ordem (desde que não tenham sido desativadas), e enviam os valores para um processador que irá analisar os valores obtidos. Em seguida, o programa espera um determinado número de segundos e inicia o ciclo novamente. Este ciclo encontra-se descrito na figura 5.3.

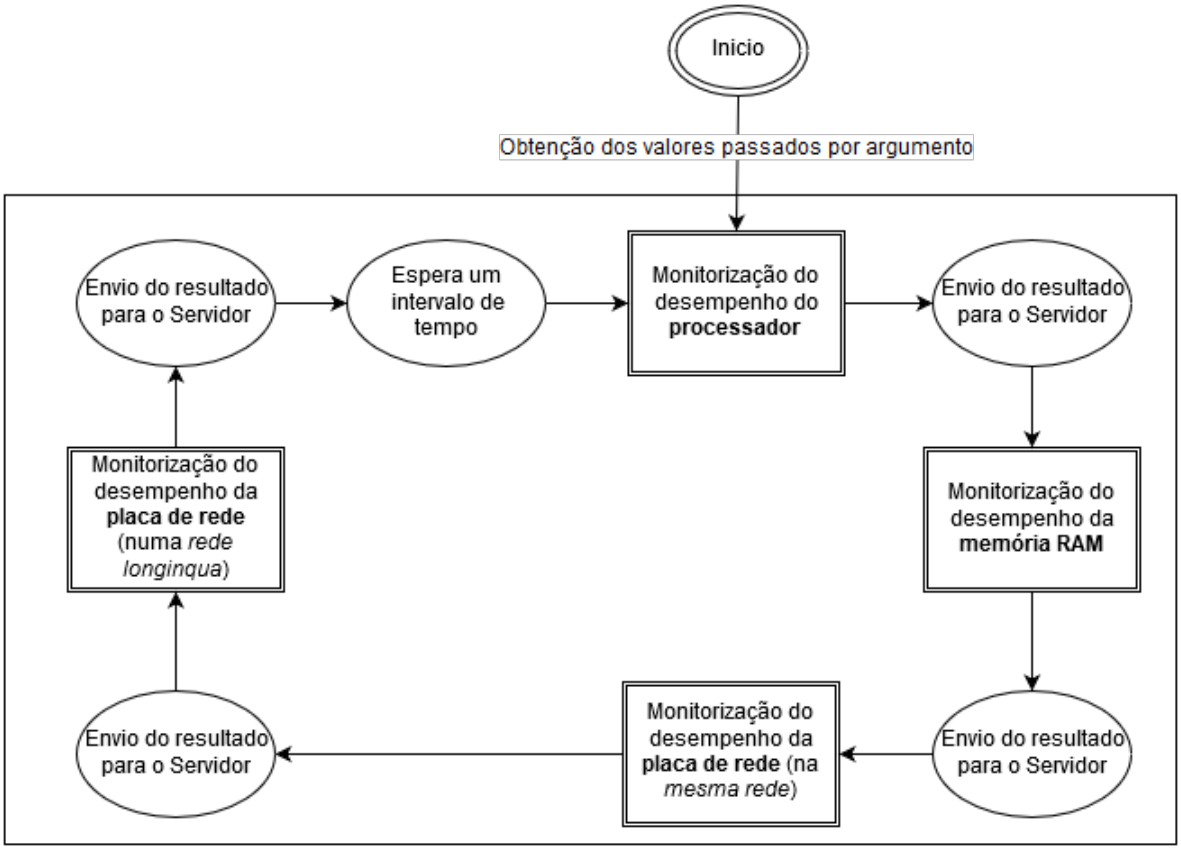


Figura 5.3: Ciclo de vida do programa de monitorização da disponibilidade de recursos.

Os valores enviados para o servidor são:

- *IP* do servidor analisado.
- Indicação do tipo de monitorização que foi efetuado (processador, memória *RAM* ou placa de rede)
- O número de operações que foi possível realizar no tempo estipulado.

A forma como este programa deve ser executado encontra-se descritoa na tabela 5.1. As diversas opções para a sua execução encontram-se na tabela 5.2.

```
USAGE: python VPSMonitorization.py <ServerAddress> <MachineIP> [options]

Options List:
    [-c CPU_Time_Monitorization(seconds)]
    [-r RAM_Memory_Time_Monitorization(seconds)]
    [-nt NET_Time_Monitorization(seconds)]
    [-s Sleep_Time_Between_Monitorization(seconds)]
    [-n Network_Monitorization_Hostname_Same_Network(IP_or_Address)]
    [-N Network_Monitorization_Hostname_Global(IP_or_Address)]
    [-dc Disable_CPU_Monitorization]
    [-dr Disable_RAM_Memory_Monitorization]
    [-dn Disable_Network_Monitorization]
```

Tabela 5.1: Método de executar o programa de monitorização da disponibilidade de recursos.

Opção	**Descrição**	**Valor por defeito**
ServerAddress	Endereço do servidor para onde os resultados serão enviados.	
MachineIP	Endereço *IP* da maquina onde se pretende realizar a monitorização.	
CPU Time Monitorization	Número de segundos que o programa de monitorização deve ser executado para cada um dos testes de processador.	5 segundos.
RAM Memory Time Monitorization	Número de segundos que o programa de monitorização deve ser executado para cada um dos testes de memória *RAM*.	5 segundos.
NET Time Monitorization	Número de segundos que o programa de monitorização deve ser executado para cada um dos testes de rede.	5 segundos
Sleep Time Between Monitorization	Número de segundos que o programa de monitorização deve ser executado para cada grupo de testes de monitorização.	50 segundos.
Network Monitorization Hostname Same Network	Endereço para uma máquina que se encontre na mesma rede que a máquina.	*Gateway* da máquina.
Network Monitorization Hostname Global	Endereço para uma máquina que se encontre preferencialmente numa localização longínqua da máquina que se pretende fazer a monitorização geograficamente.	Endereço *IP* 62.28.26.116 (*cmjornal.xl.pt*).
Disable CPU Monitorization	Desativa a monitorização do processador.	Ativo.
Disable RAM Memory Monitorization	Desativa a monitorização da memória *RAM*.	Ativo.
Disable Network Monitorization	Desativa a monitorização da rede.	Ativo.

Tabela 5.2: Detalhes das opções do programa de monitorização da disponibilidade de recursos.

5.4.2 Programa de previsão da indisponibilidade de recursos

Depois de realizados todos os testes referidos no capitulo 4, foi criado um programa com o propósito de realizar a previsão da indisponibilidade de recursos nos servidores. Este programa foi integrado no servidor para onde o programa de monitorização descrito na secção 5.4.1 envia os seus valores.

Este servidor recebe e armazena os valores, por cada um dos valores de monitorização de cada servidor que estiver a ser monitorizado. Quando recebe um novo valor o servidor, vai aplicar as fórmulas de regressão descritas para os últimos valores recebidos, obtendo uma equação, e seleciona aquela que obtiver um melhor valor de coeficiente de determinação. Em seguida, utiliza essa equação para determinar quais são os prováveis valores seguintes. Esses valores são utilizados para prever várias situações como: se o servidor vai entrar em falha, se o servidor vai voltar a entrar em bom funcionamento, ou se o servidor irá manter o seu estado atual. Este processo encontra-se demonstrado na figura 5.4.

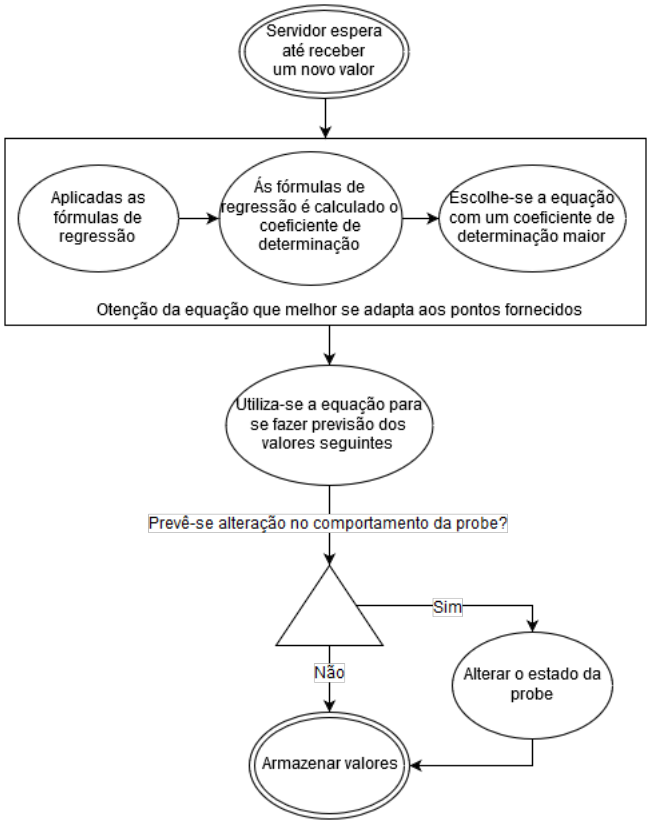


Figura 5.4: Ciclo de vida do programa de previsão de indisponibilidade de recursos.

Localização	**Descrição**	**Especificações**
Frankfurt, Alemanha	Servidor principal, onde foi colocado o servidor *Django* com o programa de previsão da indisponibilidade de recursos	Processador com 2 cores, 1GB de memória RAM, 500MB de largura de banda, a correr em *OpenVZ* e sistema operativo *Ubuntu 14.04* de *64Bit*
Londres, Inglaterra	*Probe* de teste, com o programa de monitorização	Processador com 1 cores, 512MB de memória RAM, 250MB de largura de banda, a correr em *KVM* e sistema operativo *Debian 8 (Jessie)* de *64Bit*
Suécia	*Probe* de teste, com o programa de monitorização	Processador com 1 cores, 512MB de memória RAM, 250MB de largura de banda, a correr em *KVM* e sistema operativo *Debian 8 (Jessie)* de *64Bit*
Moscovo, Rússia	*Probe* de teste, com o programa de monitorização	Processador com 1 cores, 512MB de memória RAM, 250MB de largura de banda, a correr em *KVM* e sistema operativo *Debian 8 (Jessie)* de *64Bit*
Amesterdão, Países Baixos	*Probe* de teste, com o programa de monitorização	Processador com 1 cores, 512MB de memória RAM, 250MB de largura de banda, a correr em *KVM* e sistema operativo *Ubuntu 14.04* de *64Bit*
Milão, Itália	*Probe* de teste, com o programa de monitorização	Processador com 1 cores, 512MB de memória RAM, 250MB de largura de banda, a correr em *KVM* e sistema operativo *Ubuntu 14.04* de *64Bit*
Islândia	*Probe* de teste, com o programa de monitorização	Processador com 1 cores, 512MB de memória RAM, 250MB de largura de banda, a correr em *KVM* e sistema operativo *Debian 8 (Jessie)* de *64Bit*
Israel	*Probe* de teste, com o programa de monitorização	Processador com 1 cores, 512MB de memória RAM, 250MB de largura de banda, a correr em *KVM* e sistema operativo *Debian 8 (Jessie)* de *64Bit*
Hong Kong, China	*Probe* de teste, com o programa de monitorização	Processador com 1 cores, 512MB de memória RAM, 250MB de largura de banda, a correr em *KVM* e sistema operativo *Debian 8 (Jessie)* de *64Bit*
Madrid, Espanha	*Probe* de teste, com o programa de monitorização	Processador com 1 cores, 512MB de memória RAM, 250MB de largura de banda, a correr em *KVM* e sistema operativo *Debian 8 (Jessie)* de *64Bit*
Chile	*Probe* de teste, com o programa de monitorização	Processador com 1 cores, 512MB de memória RAM, 250MB de largura de banda, a correr em *KVM* e sistema operativo *Debian 8 (Jessie)* de *64Bit*

Tabela 5.3: Descrição dos servidores de teste.

Quando se prevê que o servidor irá entrar em falha, é verificado se a *probe* em questão tem alguma outra em *backup*. Em caso positivo, essa *probe* é ativada para que se reduza a carga do servidor. Neste programa, é considerado que um servidor vai entrar em falha, quando o valor de previsão, para uma das seguintes três medições, será inferior a dois terços do melhor valor de monitorização já observado.

5.5 Resultados

De forma a testar os programas e a sua respetiva cooperação, estes foram implementados e executados em onze servidores distintos, descritos na tabela 5.3.

Os servidores foram colocados a correr durante dezasseis horas. Cada uma das *probes* executava o programa de monitorização e enviava os valores para o servidor de *Frankfurt* que analisava os dados. Entre todos valores enviados distinguem-se alguns casos especiais:

1. Quando se recebe um valor de uma *probe* do qual ainda não se tinha recebido nenhum valor de monitorização.
2. Quando se recebe um valor de monitorização superior ao atualmente guardado para um mesmo tipo de monitorização.
3. Quando o servidor se encontra ativo e se prevê a falha do servidor.
4. Quando o servidor se encontra no modo *backup* e se prevê o correto funcionamento do servidor.

Dos exemplos descritos na tabela 5.5 foi escolhido o servidor de Madrid, para as se demonstrar os exemplos. As imagens 5.5, 5.6, 5.7 e 5.8 representam cada um dos exemplos acima numerados, respetivamente, obtidos do servidor *Django*.

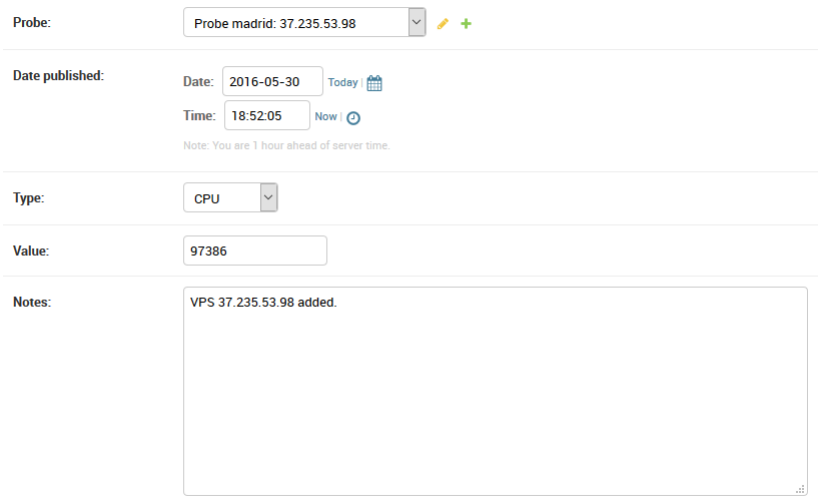


Figura 5.5: Valor de uma *probe* do qual ainda não se tinha recebido nenhum nenhum valor de monitorização.

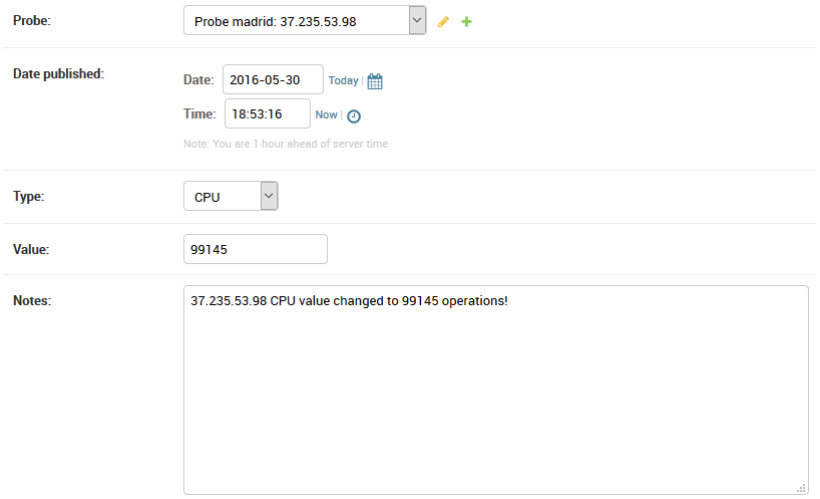


Figura 5.6: Valor de monitorização superior ao atualmente guardado para monitorização do processador.

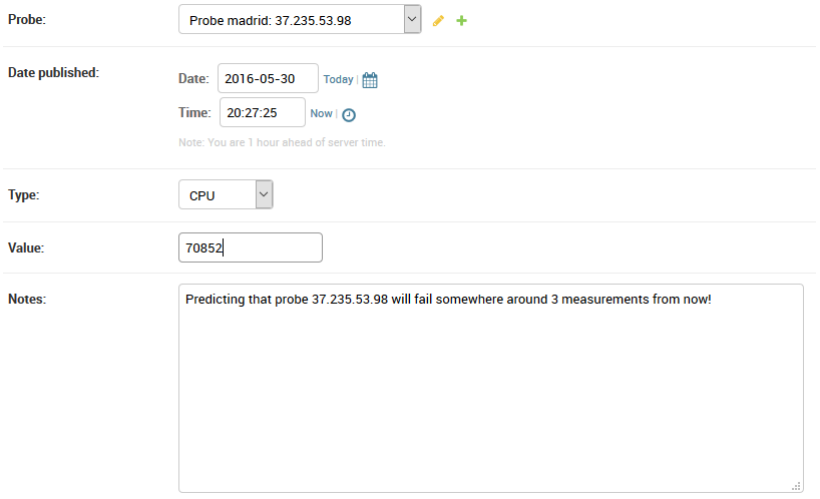


Figura 5.7: Previsão da indisponibilidade de recursos.

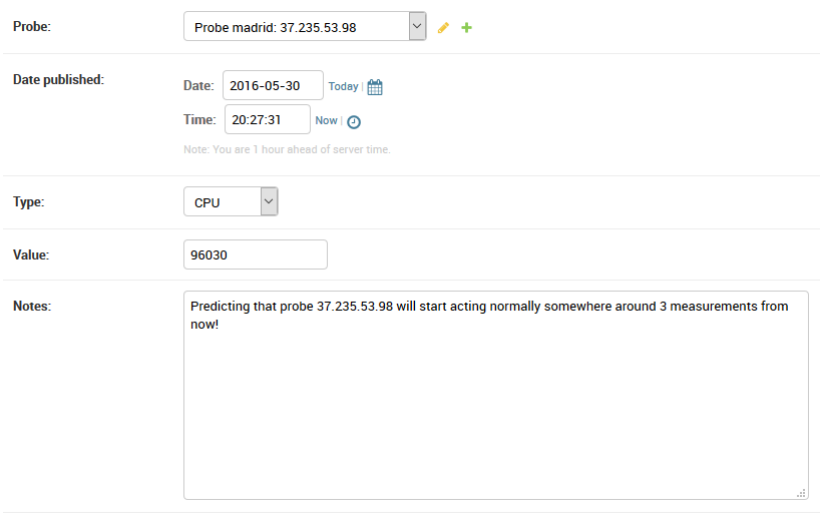
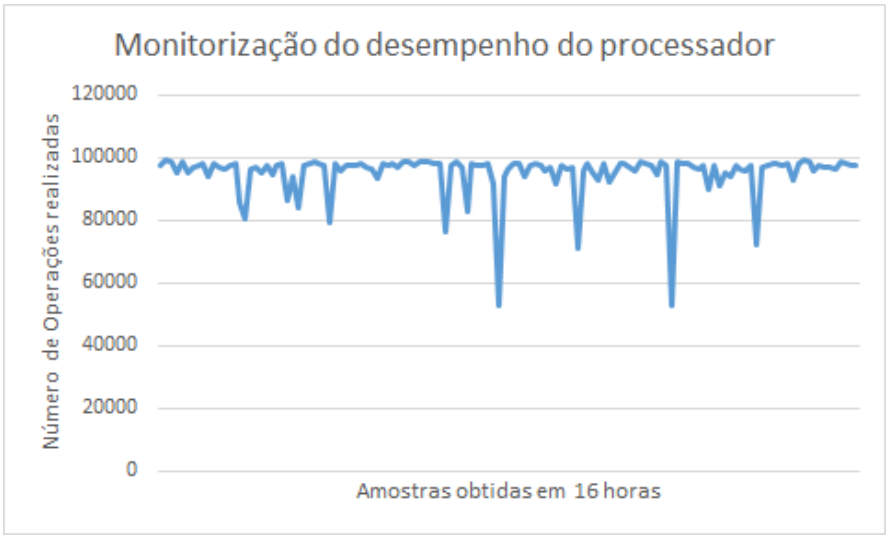
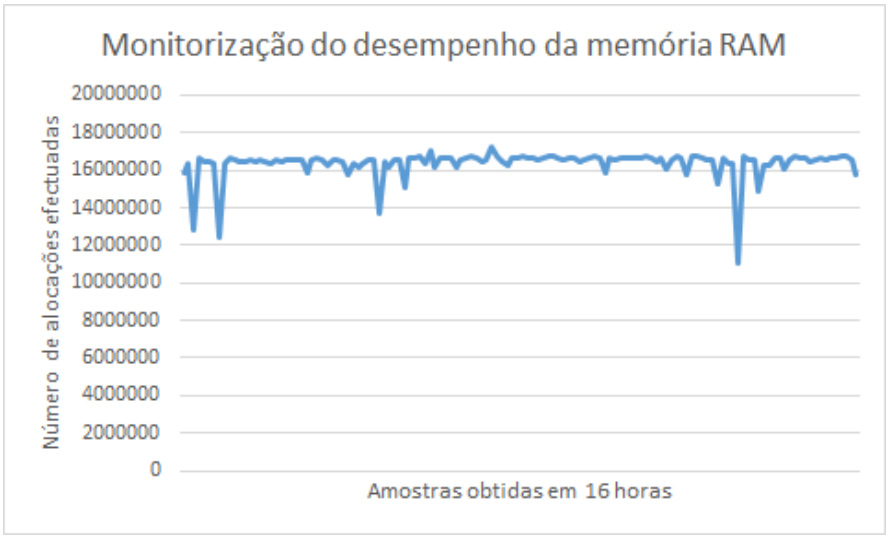


Figura 5.8: Previsão de comportamento normal.

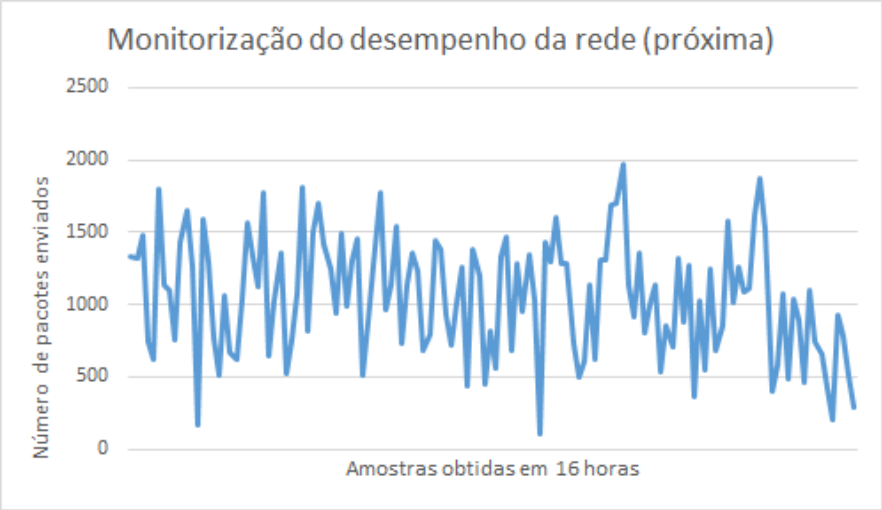
De todos os valores de monitorização foram obtidos os gráficos 5.9a, 5.9b, 5.9c e 5.9d que correspondem à variação no número de operações realizadas pelo processador, memória *RAM*, placa de rede para uma rede próxima e placa de rede para uma rede longínqua. Os gráficos dos restantes servidores encontram-se no anexo G.



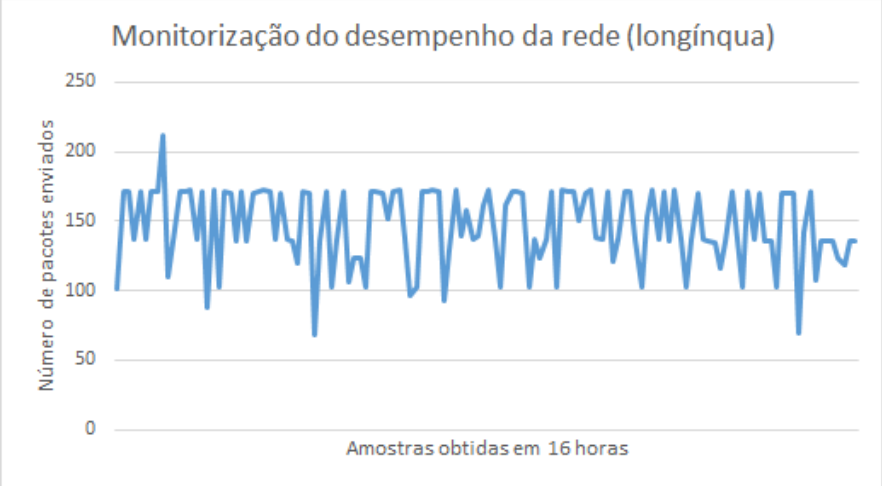
(a) Monitorização do processador da *probe* de *Madrid*.



(b) Monitorização da memória *RAM* da *probe* de *Madrid*.



(c) Monitorização da placa de rede para redes próximas da *probe* de *Madrid*.



(d) Monitorização da placa de rede para redes longínquas da *probe* de *Madrid*.

Figura 5.9: Representações dos resultados da monitorização da *probe* de *Madrid*.

6 | Conclusão

Nesta dissertação foram apresentados os passos necessários para se realizar um sistema de monitorização da disponibilidade de recursos e previsão da indisponibilidade de recursos.

Como foi verificado, devido ao facto de que cada cliente não saber as especificações concretas do servidor no qual está alojado, a monitorização de desempenho tem que ser feita de forma indireta. Esta monitorização é feita através da verificação e comparação da eficiência da componente que se quer analisar, ao longo do tempo. Neste trabalho foram analisadas várias situações que poderiam ser utilizadas para se proceder a esta monitorização, e comprovou-se a possibilidade de se fazer uma análise concreta do estado de ocupação de diversas componentes necessárias ao correto funcionamento de um servidor *cloud* (processador, memória *RAM* e conexão de rede).

Verificou-se também que, utilizando esses valores, é possível fazer uma aproximação da eficiência dessa componente num futuro próximo, através do uso de ferramentas de previsão. Essas ferramentas tornam possível que sejam aplicados métodos de proteção contra falhas, antes mesmo do servidor entrar em estado crítico. Isto permite poupar tempo e informação, assim como aumentar produtividade das empresas/instituições.

Espera-se que, com este projeto, seja possível ajudar diversas empresas, organizações e clientes/fornecedores de servidores *cloud*, a usufruir de forma mais segura e eficientes dos recursos fornecidos, reduzindo o número de falhas, perda e informação e redução de produtividade.

6.1 Trabalho Futuro

Os testes efetuados ao longo desta dissertação foram realizados num único computador. Este encontrava-se a funcionar como se de um servidor se tratasse, sendo as suas capacidades computacionais muito inferiores às dos servidores normalmente utilizados. Como trabalho futuro, deverão ser feitos testes semelhantes num servidor, de forma a confirmar se os resultados obtidos são, de facto, valores fiáveis.

Há que ter em consideração que não existem resultados comprovativos da eficiência deste sistema. Isto deveu-se ao facto de que, para verificar se um dado valor foi previsto como falha (ou não) corretamente, é necessário o acesso aos valores do servidor onde as máquinas estão instaladas. No decorrer desta dissertação, apesar de se ter acesso a diversas máquinas virtuais,

não foi possível obter os valores reais dos servidores físicos, pelo que não foi possível realizar esta análise.

Bibliografia

[1] José Luís da Silva Rosa. Customer-side detection of bgp routing attacks. 2016.

[2] Peter Mell and Tim Grance. The nist definition of cloud computing. 2011. `http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf`.

[3] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[4] Vps vs. cloud hosting. `http://www.rackspace.co.uk/cloud-computing/vps`. Acedido em Fevereiro de 2016.

[5] Rich Uhlig, Gil Neiger, Dion Rodgers, Amy L Santoni, Fernando Martins, Andrew V Anderson, Steven M Bennett, Alain Kägi, Felix H Leung, and Larry Smith. Intel virtualization technology. *Computer*, 38(5):48–56, 2005.

[6] Gerald J Popek and Robert P Goldberg. Formal requirements for virtualizable third generation architectures. *Communications of the ACM*, 17(7):412–421, 1974.

[7] NM Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862–876, 2010.

[8] Andrey Mirkin, Alexey Kuznetsov, and Kir Kolyshkin. Containers checkpointing and live migration. In *Proceedings of the Linux Symposium*, volume 2, pages 85–90, 2008.

[9] Pradeep Padala, Xiaoyun Zhu, Zhikui Wang, Sharad Singhal, Kang G Shin, et al. Performance evaluation of virtualization technologies for server consolidation. *HP Labs Tec. Report*, 2007.

[10] Openvz virtuozzo containers - network virtualization. `https://openvz.org/Features#Network_virtualization`. Acedido em Fevereiro de 2016.

[11] Charles David Graziano. A performance analysis of xen and kvm hypervisors for hosting the xen worlds project. 2011.

[12] Xen networking - virtual network interfaces. `http://wiki.xenproject.org/wiki/Xen_Networking#Virtual_Network_Interfaces`. Acedido em Fevereiro de 2016.

[13] Charles David Graziano. A performance analysis of xen and kvm hypervisors for hosting the xen worlds project. 2011.

[14] Shengzhao Li, Qinfen Hao, Limin Xiao, and Qingling Xu. Optimizing network virtualization in kernel-based virtual machine. In *Information Science and Engineering (ICISE), 2009 1st international Conference on*, pages 282–285. IEEE, 2009.

[15] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[16] Robert Inkol, Collin Wilson, and Mathieu Eidus. Applications of performance benchmarking to the development of signal processing systems based on personal computer technology. In *Electrical and Computer Engineering, 2006. CCECE'06. Canadian Conference on*, pages 41–45. IEEE, 2006.

[17] David H Bailey, Simon M Plouffe, Peter B Borwein, and Jonathan M Borwein. The quest for pi. *The Mathematical Intelligencer*, 19(1):50–56, 1997.

[18] John O Rawlings, Sastry G Pantula, and David A Dickey. *Applied regression analysis: a research tool*. Springer Science & Business Media, 1998.

[19] Christopher M Bishop. Pattern recognition. *Machine Learning*, 2006.

[20] Algorithms (polynomial regression). `http://www.originlab.com/doc/Origin-Help/PR-Algorithm`. Acedido em Maio de 2016.

[21] Polynomial regression. `http://arachnoid.com/sage/polynomial.html`. Acedido em Maio de 2016.

[22] JF Kenney and ES Keeping. Linear regression and correlation. *Mathematics of statistics*, 1:252–285, 1962.

[23] Algorithms (linear regression). `http://www.originlab.com/doc/Origin-Help/LR-Algorithm`. Acedido em Maio de 2016.

[24] Quadratic regression. `http://keisan.casio.com/exec/system/14059932254941`. Acedido em Maio de 2016.

[25] VB Melas. Optimal designs for exponential regression. *Statistics: A Journal of Theoretical and Applied Statistics*, 9(1):45–59, 1978.

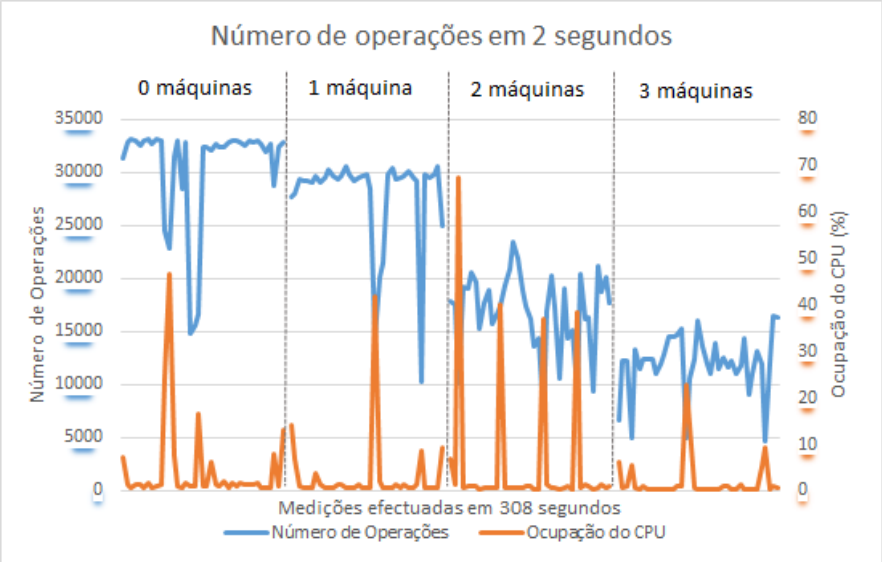
[26] Least squares fitting–exponential. `http://mathworld.wolfram.com/LeastSquaresFittingExponential.html`. Acedido em Maio de 2016.

[27] About python. `https://www.python.org/about/`. Acedido em Junho de 2016.

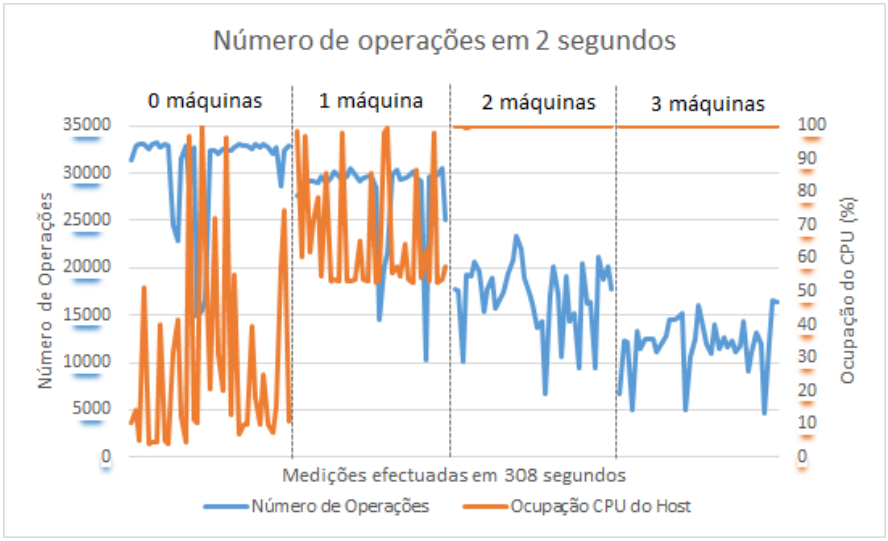
[28] Python beginners guide - overview. `https://wiki.python.org/moin/BeginnersGuide/Overview`. Acedido em Junho de 2016.

[29] Django. the web framework for perfectionists with deadlines. `https://www.djangoproject.com/`. Acedido em Junho de 2016.

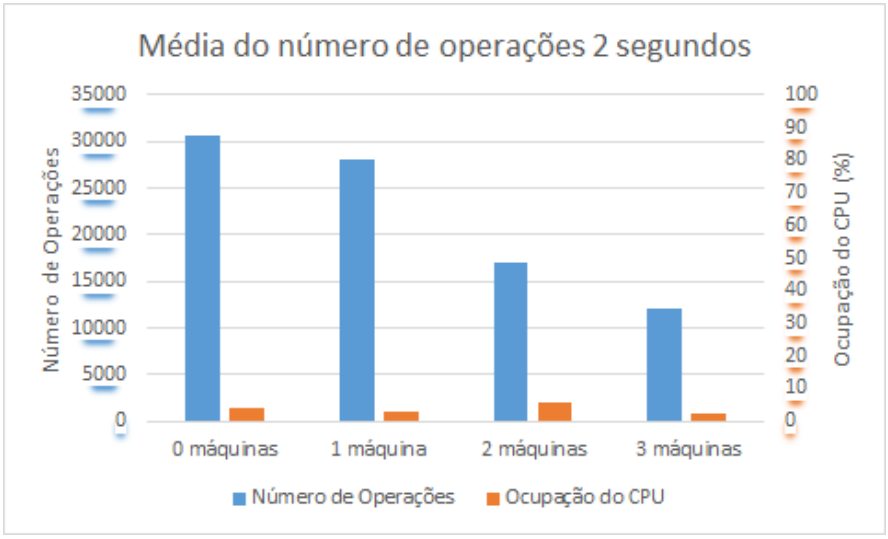
A | Resultados Completos dos Testes de Monitorização do Processador (KVM)



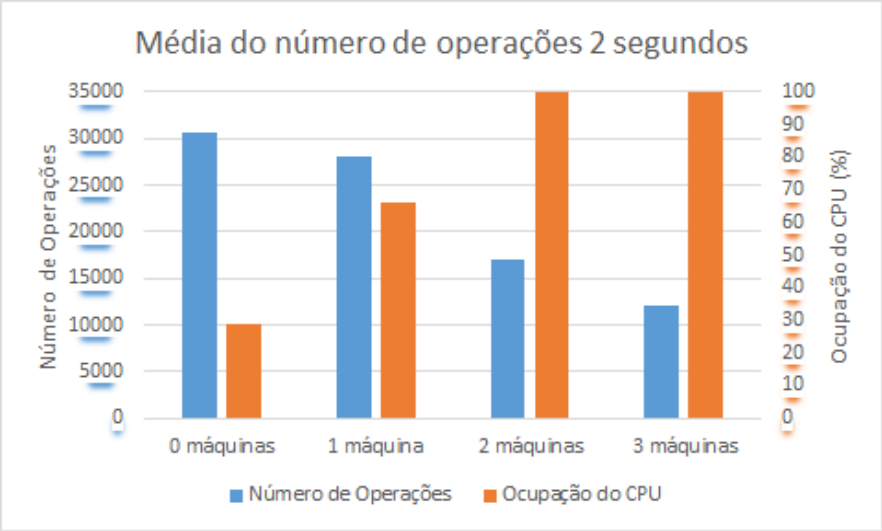
(a) Número de Operações efetuadas vs Medição de ocupação do *CPU* na máquina



(b) Número de Operações efetuadas vs Medição de ocupação do *CPU* no *Host*

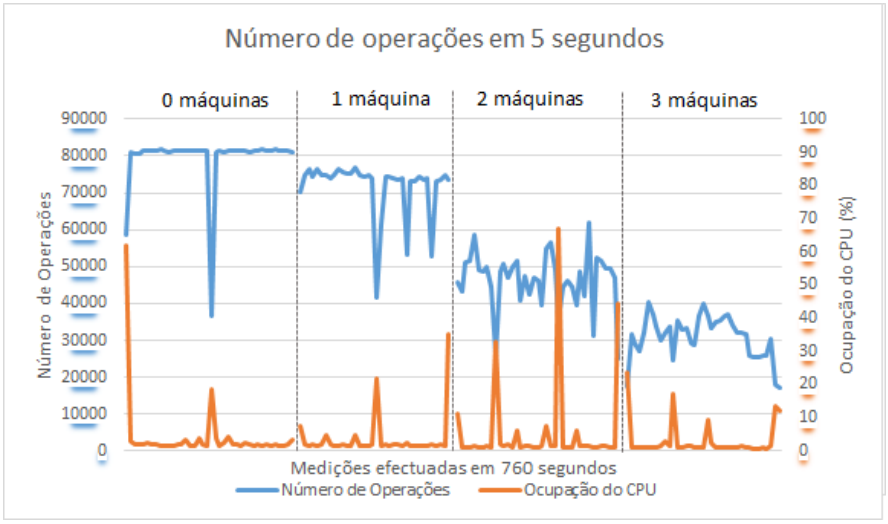


(c) Número médio de Operações efetuadas vs Média da ocupação do *CPU* na máquina

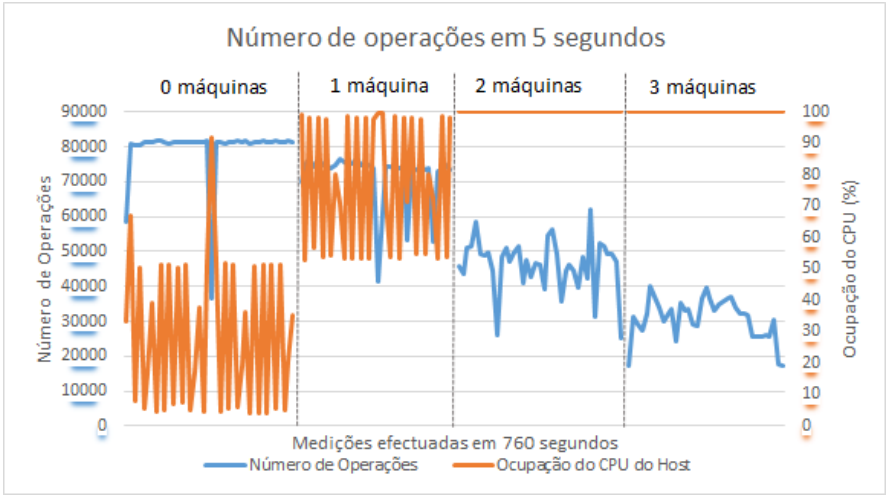


(d) Número médio de Operações efetuadas vs Média da ocupação do *CPU* no *Host*

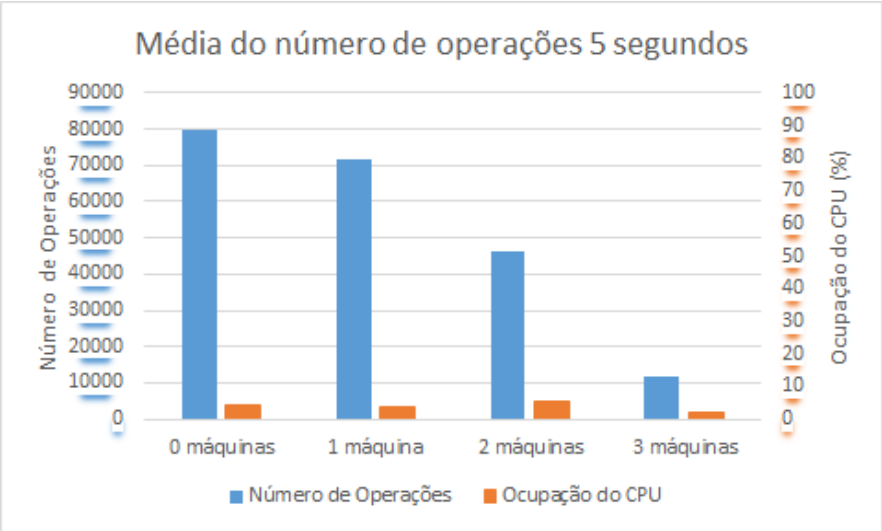
Figura A.1: Resultados obtidos com a monitorização com intervalo fixo de 2 segundos, variando a ocupação do *CPU*.



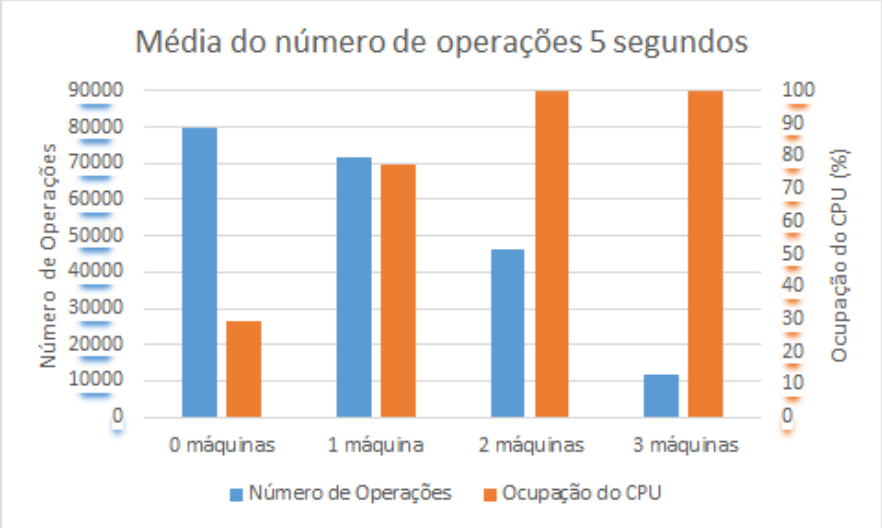
(a) Número de Operações efetuadas vs Medição de ocupação do *CPU* na máquina



(b) Número de Operações efetuadas vs Medição de ocupação do *CPU* no *Host*

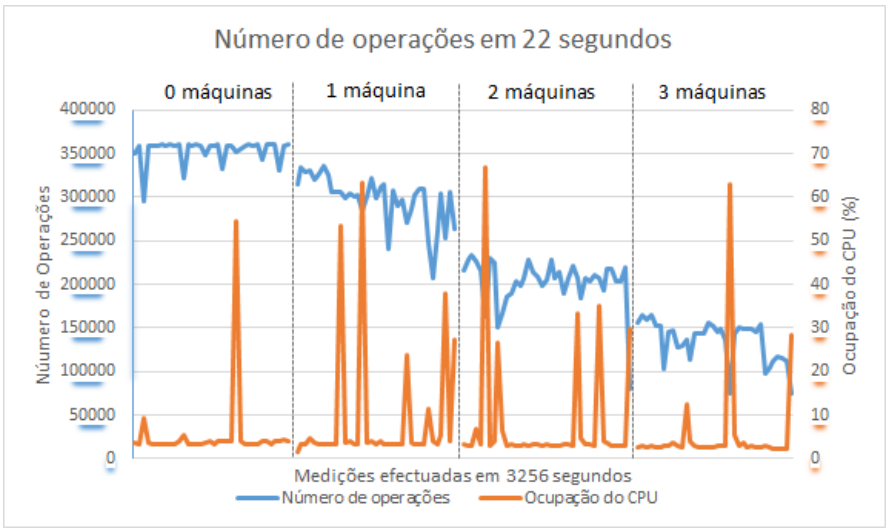


(c) Número médio de Operações efetuadas vs Média da ocupação do *CPU* na máquina

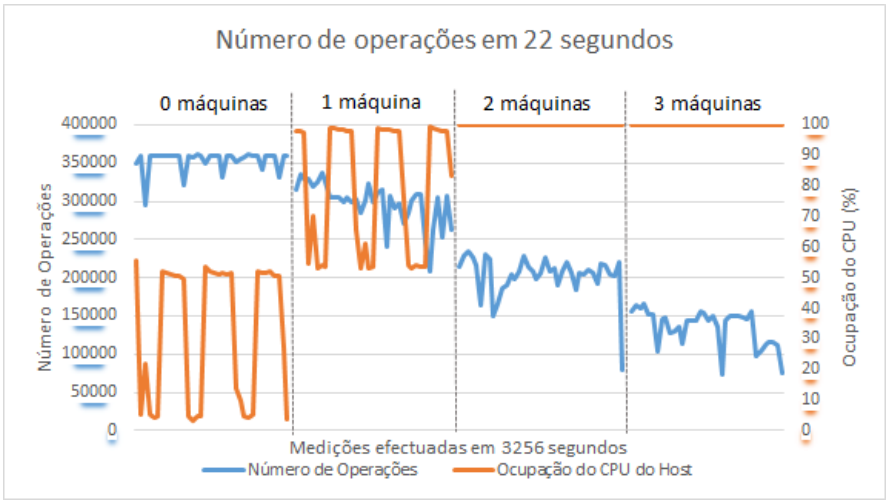


(d) Número médio de Operações efetuadas vs Média da ocupação do *CPU* no *Host*

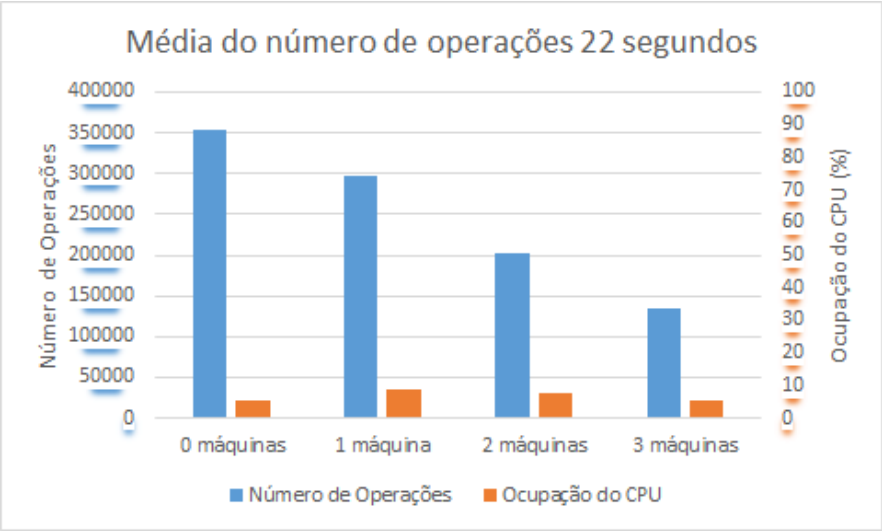
Figura A.2: Resultados obtidos com a monitorização com intervalo fixo de 5 segundos, variando a ocupação do *CPU*.



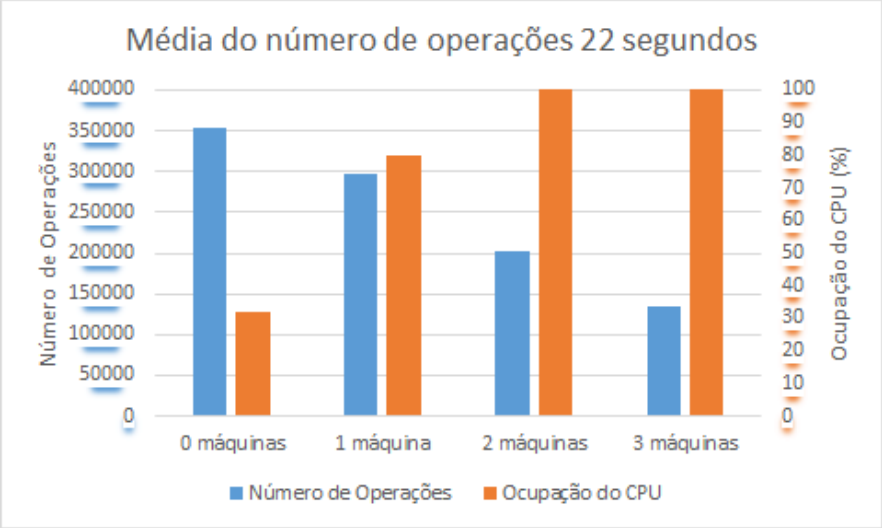
(a) Número de Operações efetuadas vs Medição de ocupação do *CPU* na máquina



(b) Número de Operações efetuadas vs Medição de ocupação do *CPU* no *Host*

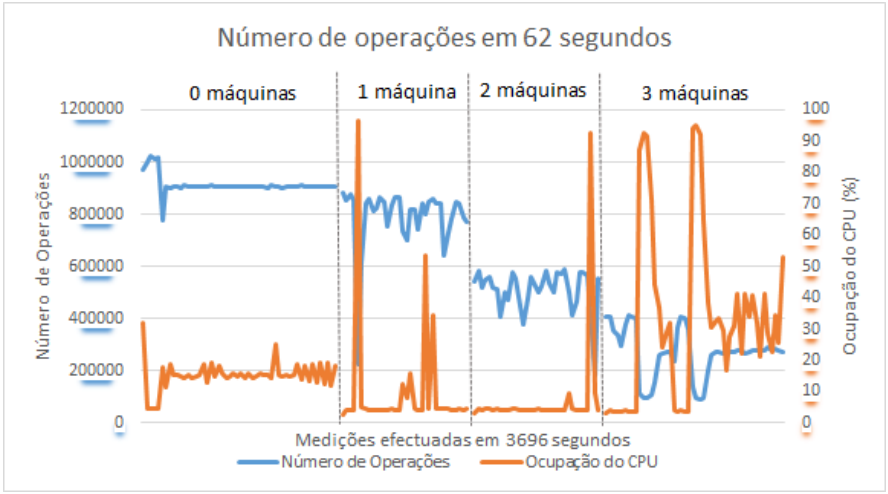


(c) Número médio de Operações efetuadas vs Média da ocupação do *CPU* na máquina

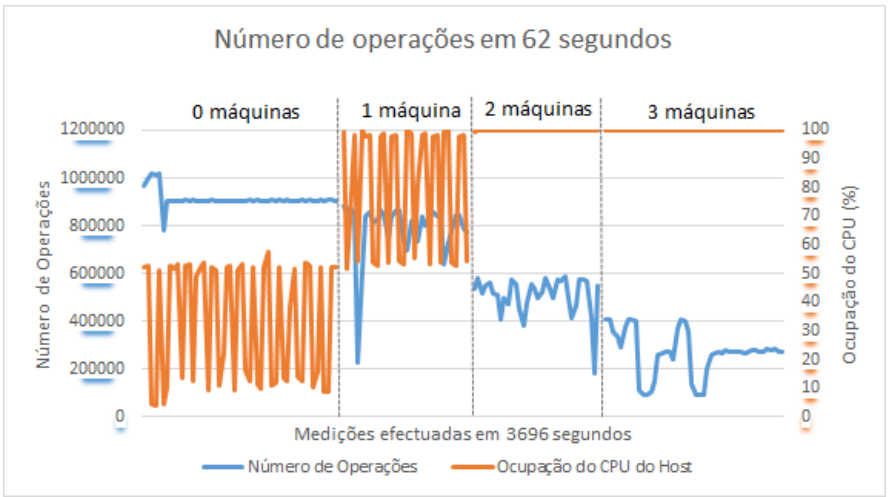


(d) Número médio de Operações efetuadas vs Média da ocupação do *CPU* no *Host*

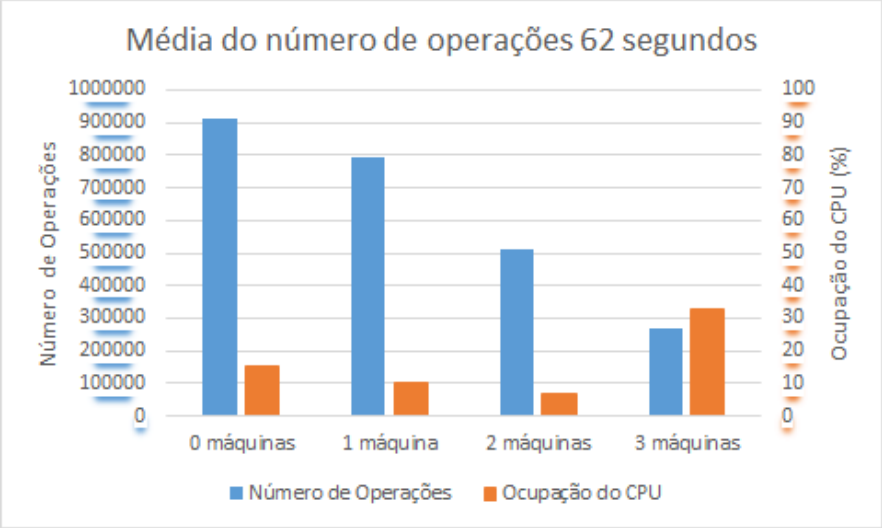
Figura A.3: Resultados obtidos com a monitorização com intervalo fixo de 22 segundos, variando a ocupação do *CPU*.



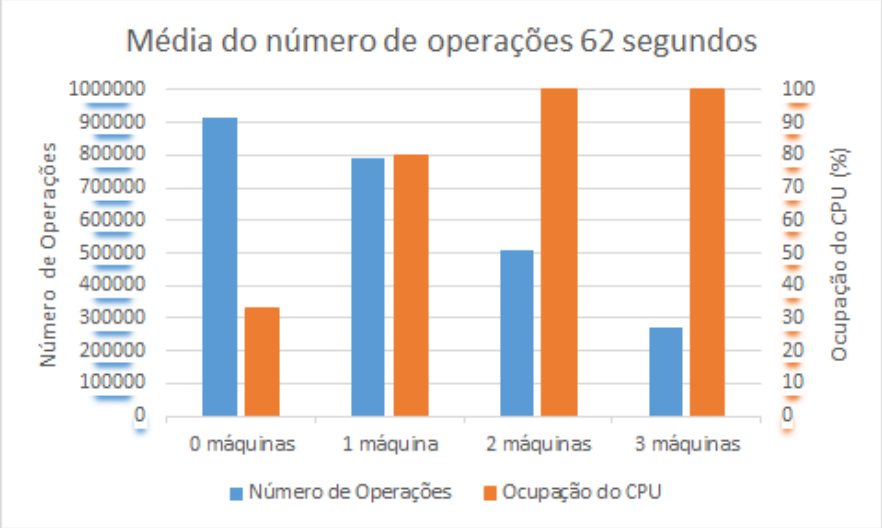
(a) Número de Operações efetuadas vs Medição de ocupação do *CPU* na máquina



(b) Número de Operações efetuadas vs Medição de ocupação do *CPU* no *Host*



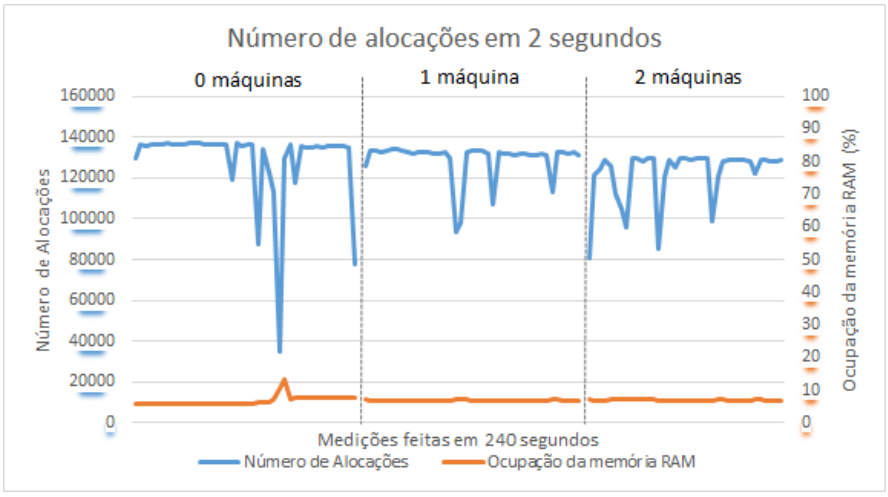
(c) Número médio de Operações efetuadas vs Média da ocupação do *CPU* na máquina



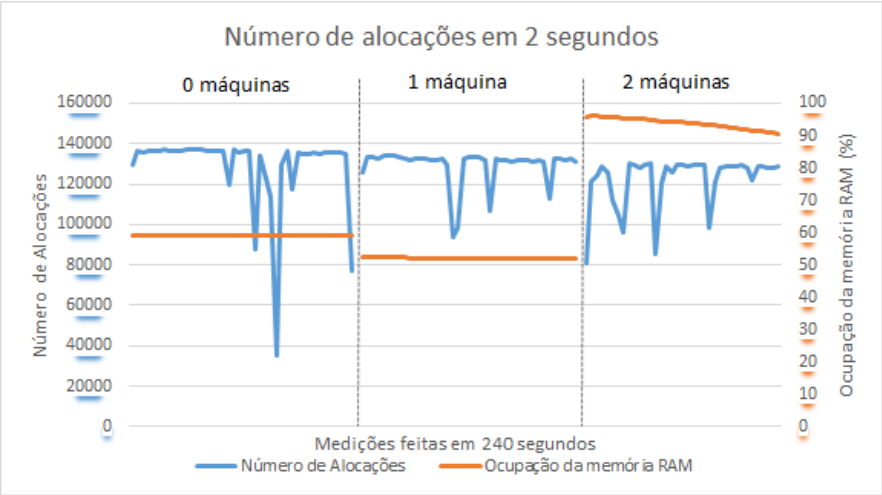
(d) Número médio de Operações efetuadas vs Média da ocupação do *CPU* no *Host*

Figura A.4: Resultados obtidos com a monitorização com intervalo fixo de 62 segundos, variando a ocupação do *CPU*.

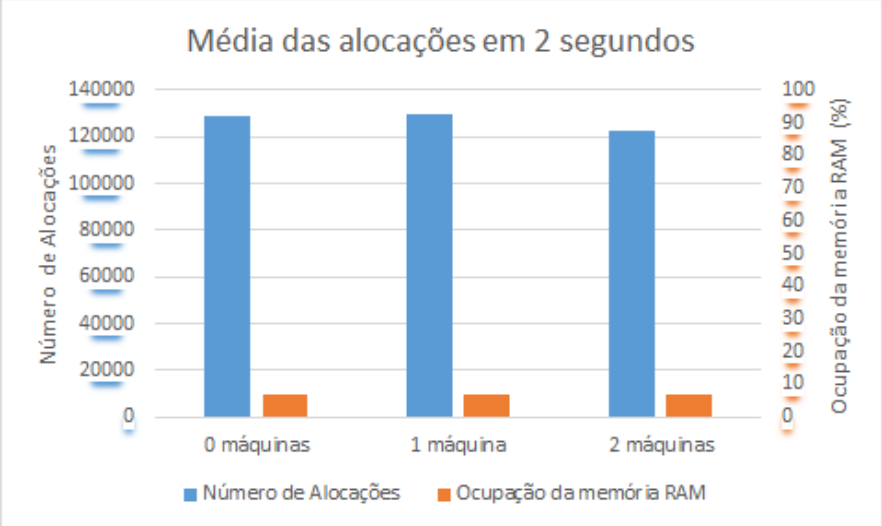
B | Resultados Completos dos Testes de Monitorização da Memória RAM (KVM)



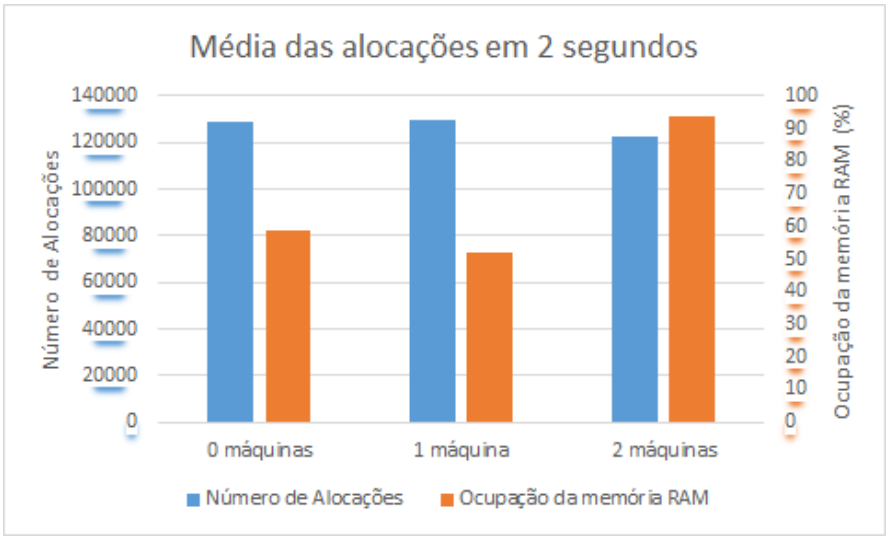
(a) Número de Operações efetuadas vs Medição de ocupação da memória *RAM* na máquina



(b) Número de Operações efetuadas vs Medição de ocupação da memória *RAM* no *Host*

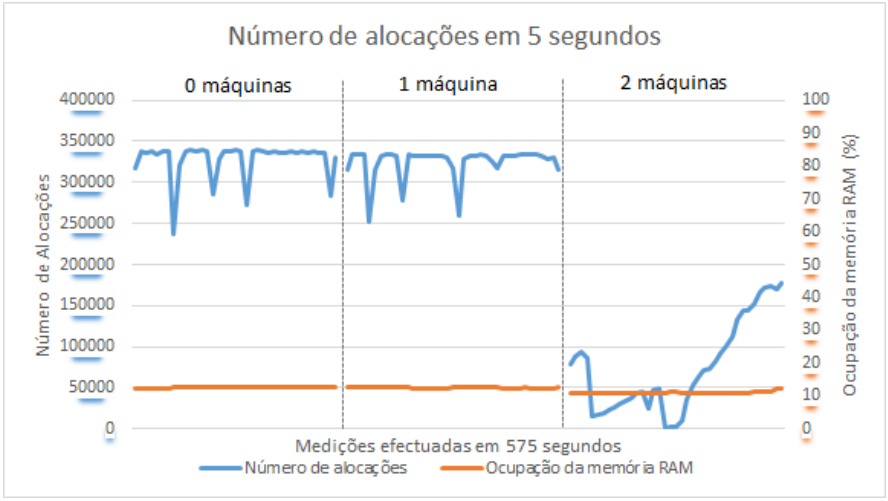


(c) Número médio de Operações efetuadas vs Média da ocupação da memória *RAM* na máquina

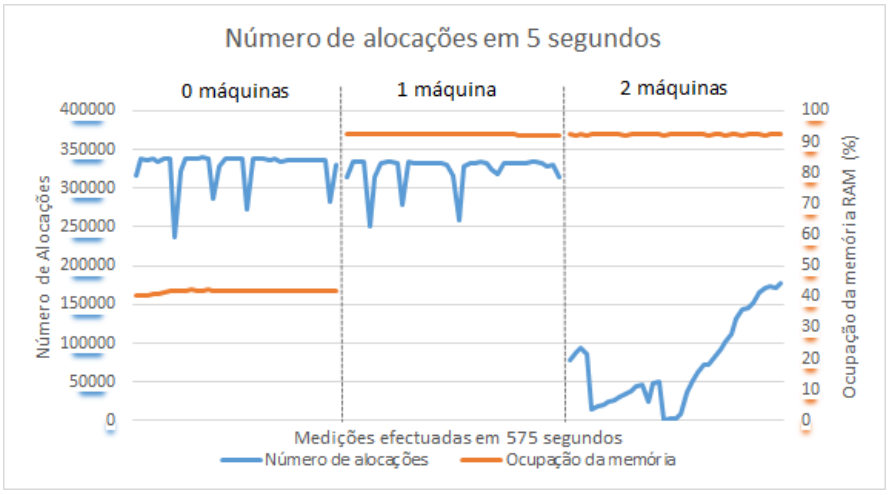


(d) Número médio de Operações efetuadas vs Média da ocupação da memória *RAM* no *Host*

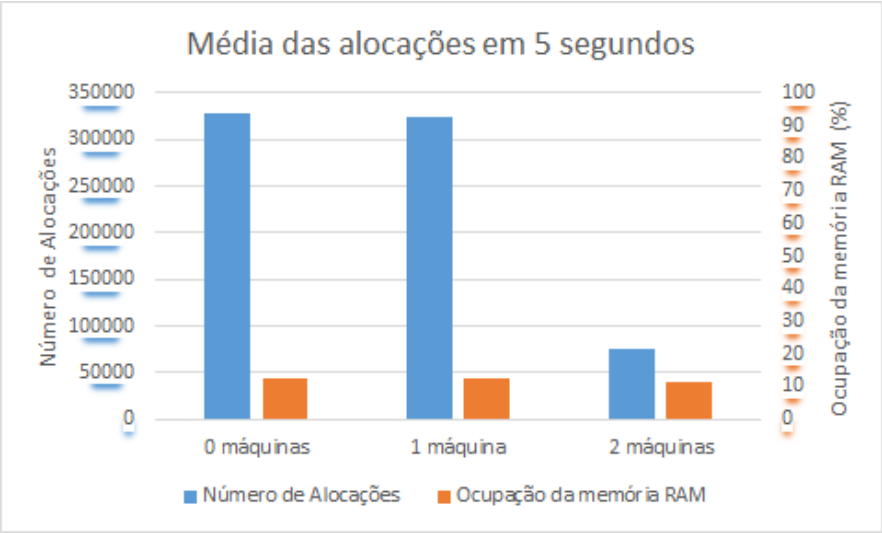
Figura B.1: Resultados obtidos com a monitorização com intervalo fixo de 2 segundos, variando a ocupação da memória *RAM*.



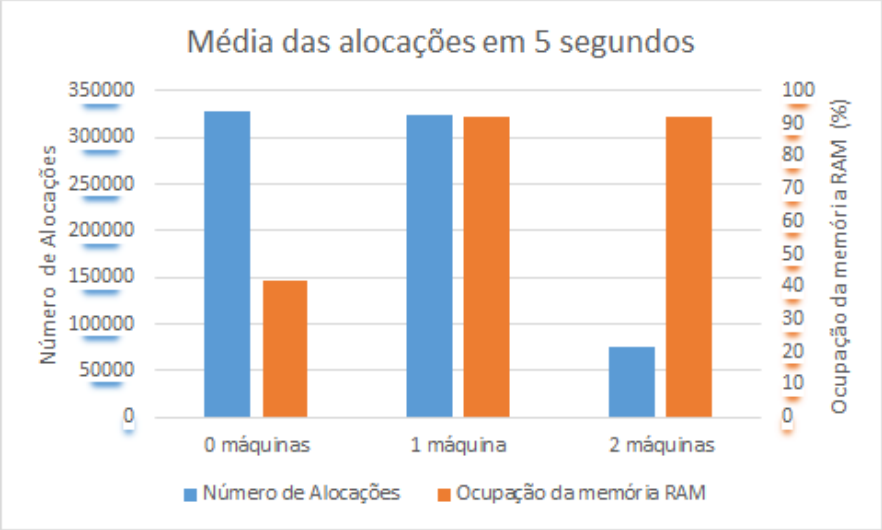
(a) Número de Operações efetuadas vs Medição de ocupação da memória *RAM* na máquina



(b) Número de Operações efetuadas vs Medição de ocupação da memória *RAM* no *Host*

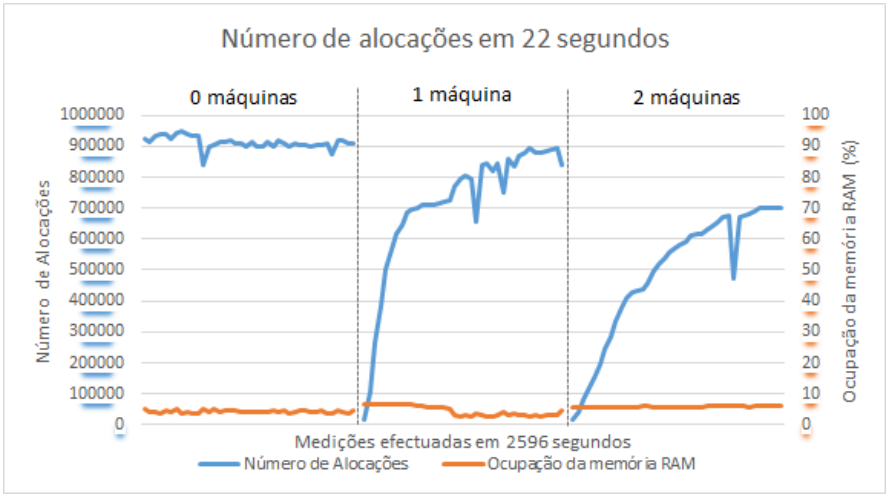


(c) Número médio de Operações efetuadas vs Média da ocupação da memória *RAM* na máquina

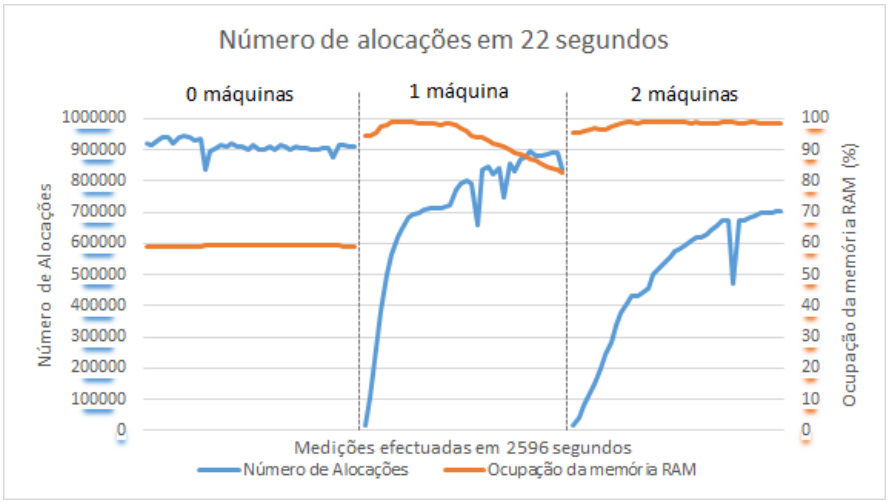


(d) Número médio de Operações efetuadas vs Média da ocupação da memória *RAM* no *Host*

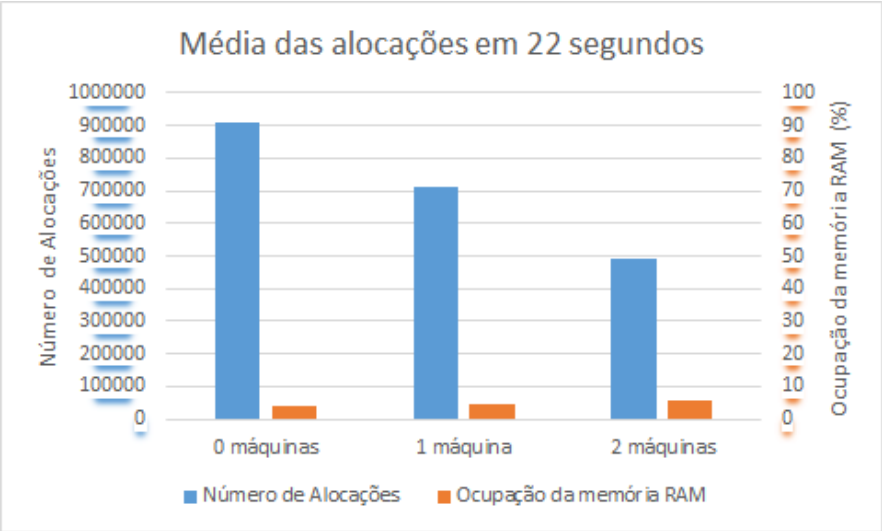
Figura B.2: Resultados obtidos com a monitorização com intervalo fixo de 5 segundos, variando a ocupação da memória *RAM*.



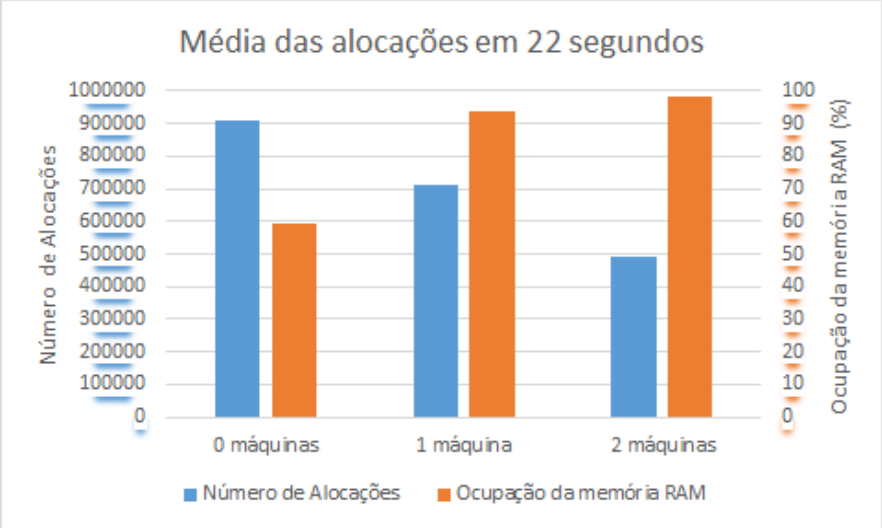
(a) Número de Operações efetuadas vs Medição de ocupação da memória *RAM* na máquina



(b) Número de Operações efetuadas vs Medição de ocupação da memória *RAM* no *Host*



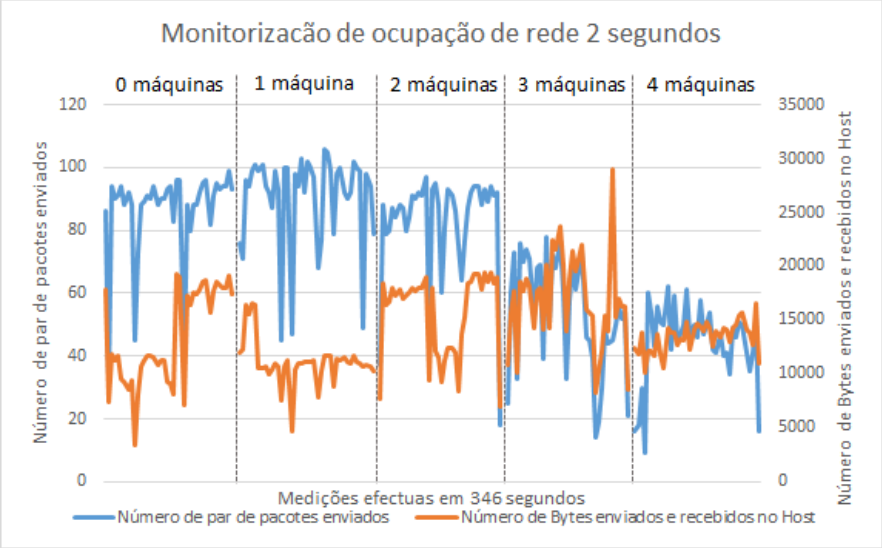
(c) Número médio de Operações efetuadas vs Média da ocupação da memória *RAM* na máquina



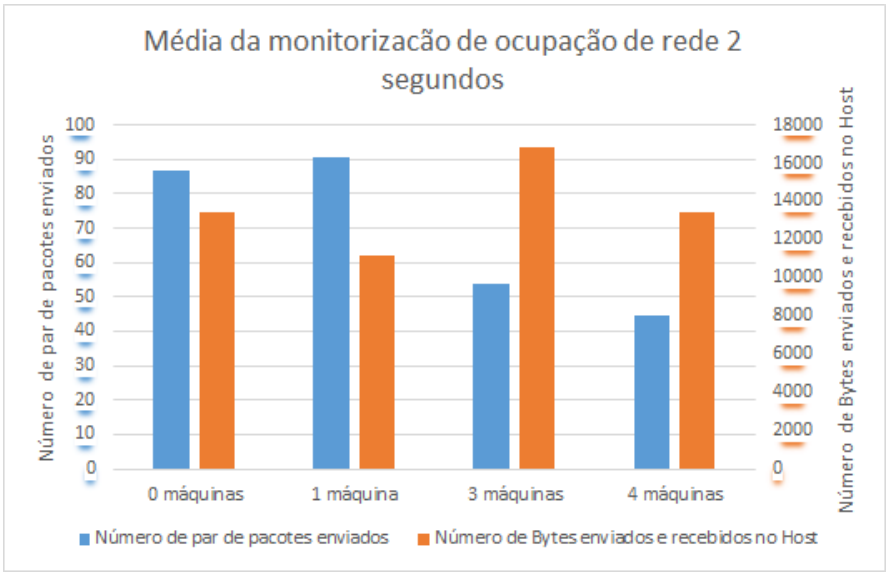
(d) Número médio de Operações efetuadas vs Média da ocupação da memória *RAM* no *Host*

Figura B.3: Resultados obtidos com a monitorização com intervalo fixo de 22 segundos, variando a ocupação da memória *RAM*.

C | Resultados Completos dos Testes de Monitorização da rede (KVM)

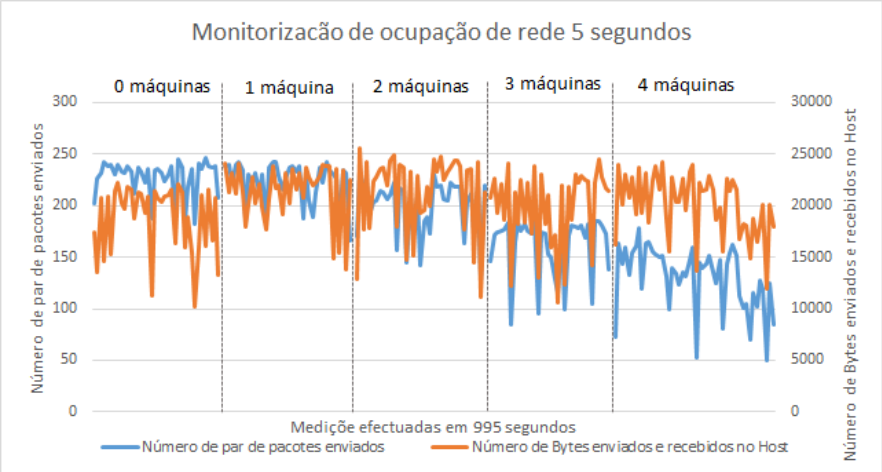


(a) Número de par de pacotes enviados vs Número de Bytes enviados e recebidos no *Host*

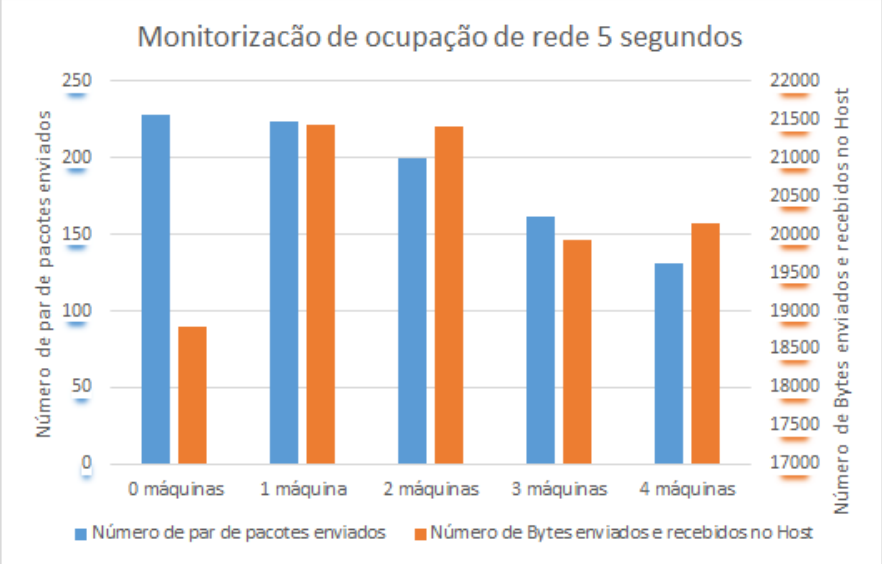


(b) Média do número de par de pacotes enviados vs Média número de Bytes enviados e recebidos no *Host*

Figura C.1: Resultados obtidos com a monitorização com intervalo fixo de 2 segundos, variando a ocupação da rede.

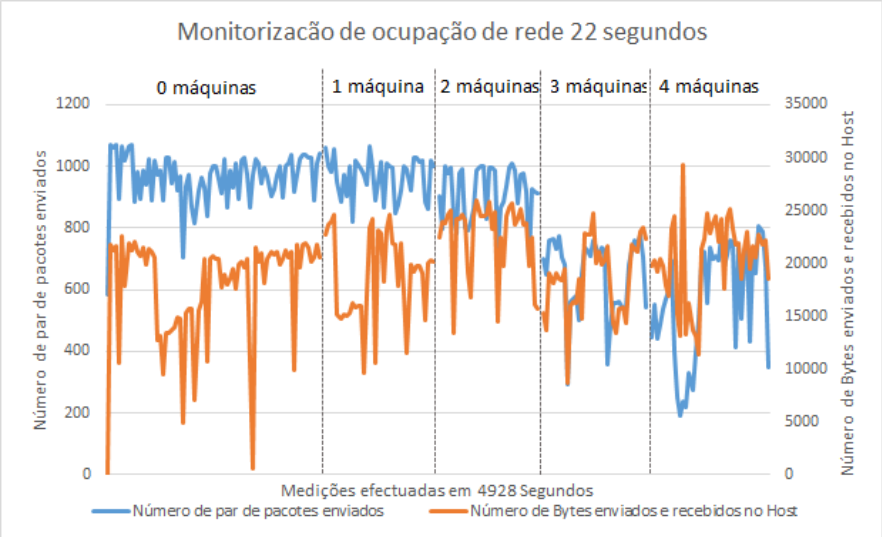


(a) Número de par de pacotes enviados vs Número de Bytes enviados e recebidos no *Host*

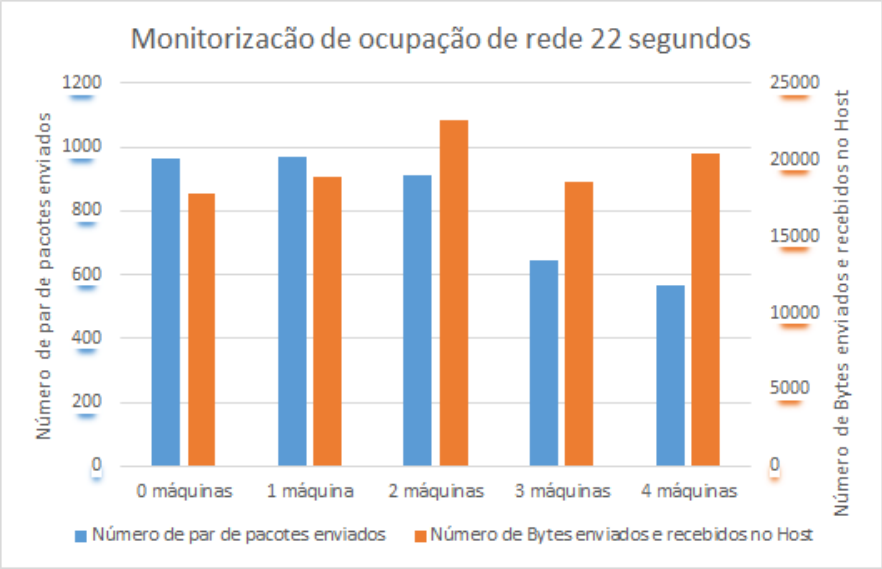


(b) Média do número de par de pacotes enviados vs Média número de Bytes enviados e recebidos no *Host*

Figura C.2: Resultados obtidos com a monitorização com intervalo fixo de 5 segundos, variando a ocupação da rede.



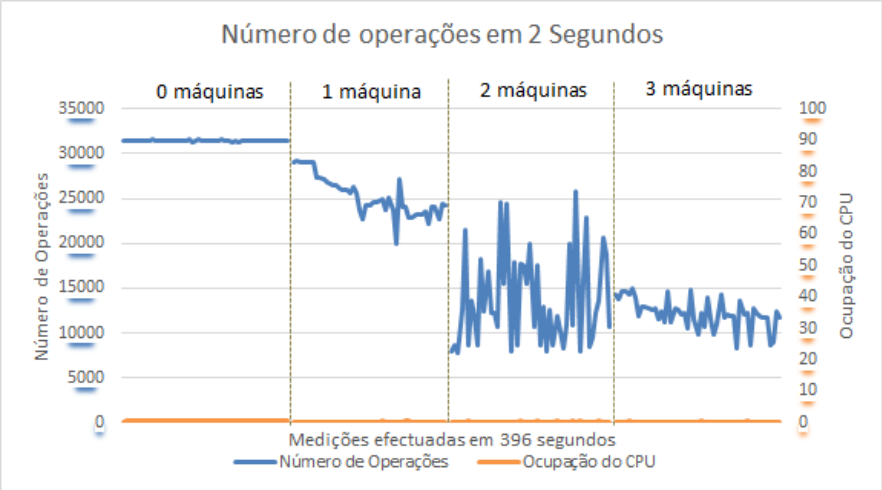
(a) Número de par de pacotes enviados vs Número de Bytes enviados e recebidos no *Host*



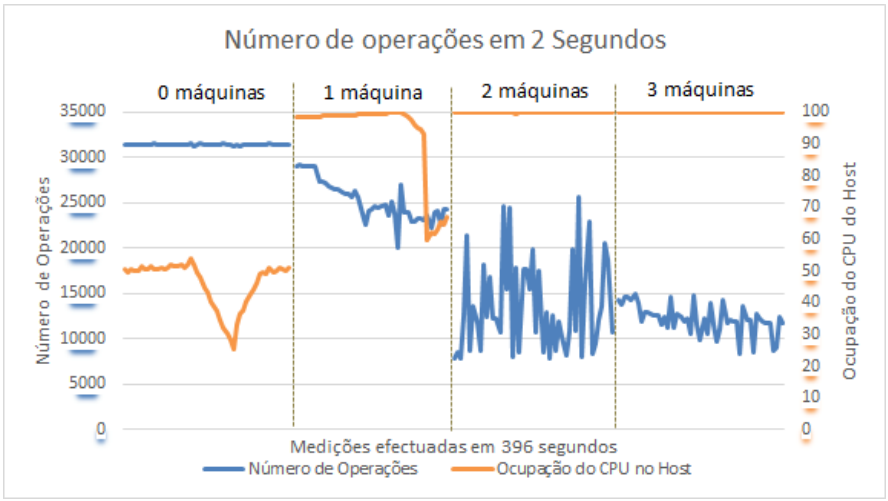
(b) Média do número de par de pacotes enviados vs Média número de Bytes enviados e recebidos no *Host*

Figura C.3: Resultados obtidos com a monitorização com intervalo fixo de 22 segundos, variando a ocupação da rede.

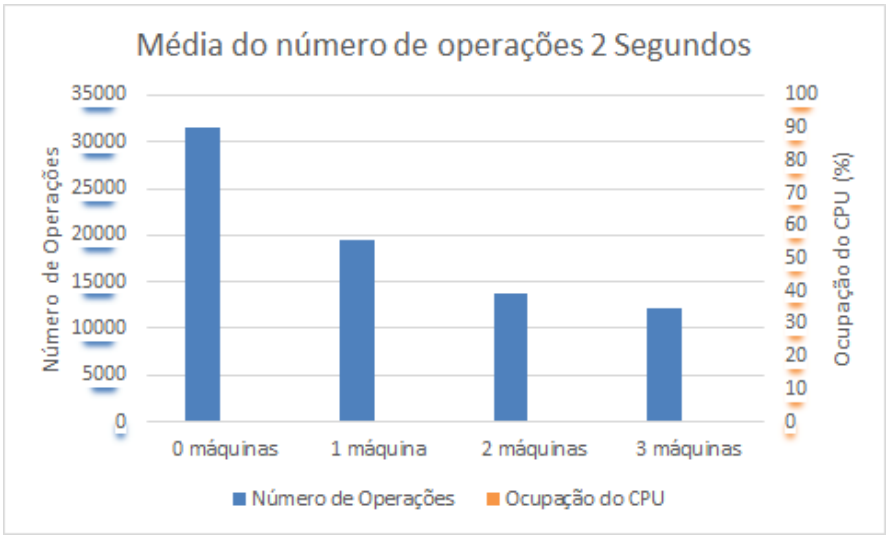
D | Resultados Completos dos Testes de Monitorização do Processador (OpenVZ)



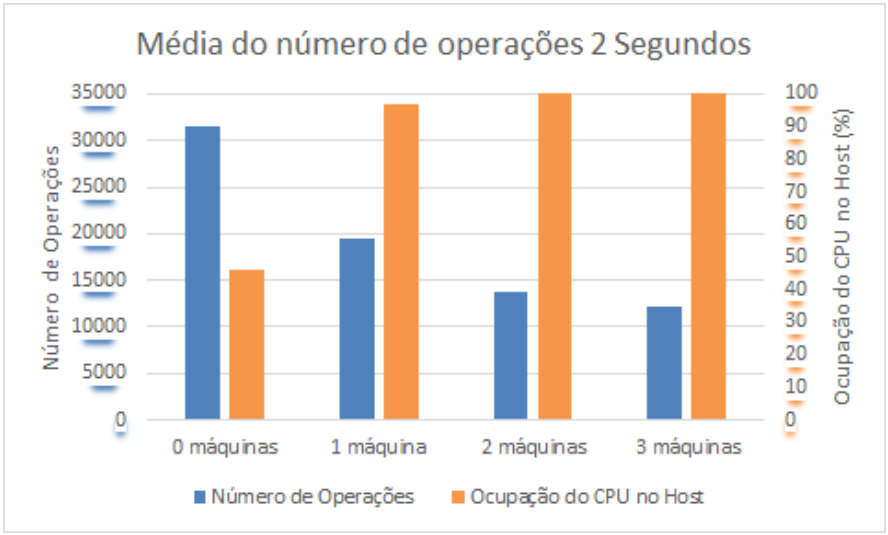
(a) Número de Operações efetuadas vs Medição de ocupação do *CPU* na máquina



(b) Número de Operações efetuadas vs Medição de ocupação do *CPU* no *Host*

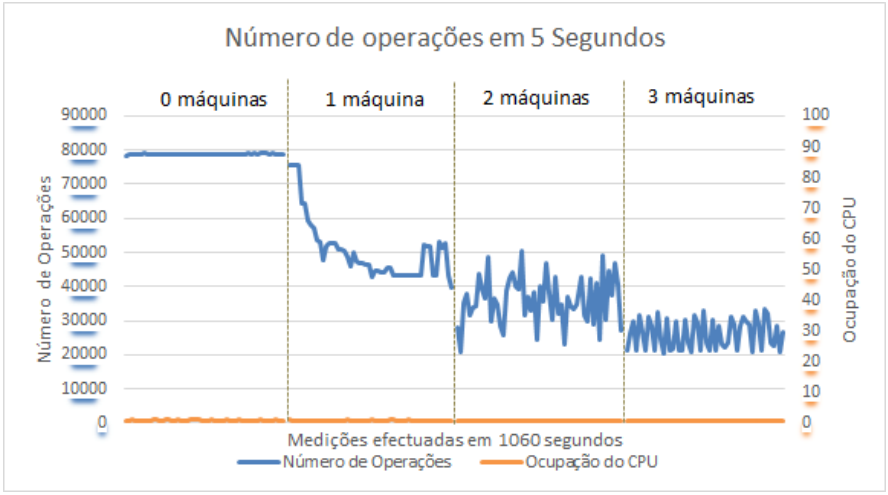


(c) Número médio de Operações efetuadas vs Média da ocupação do *CPU* na máquina

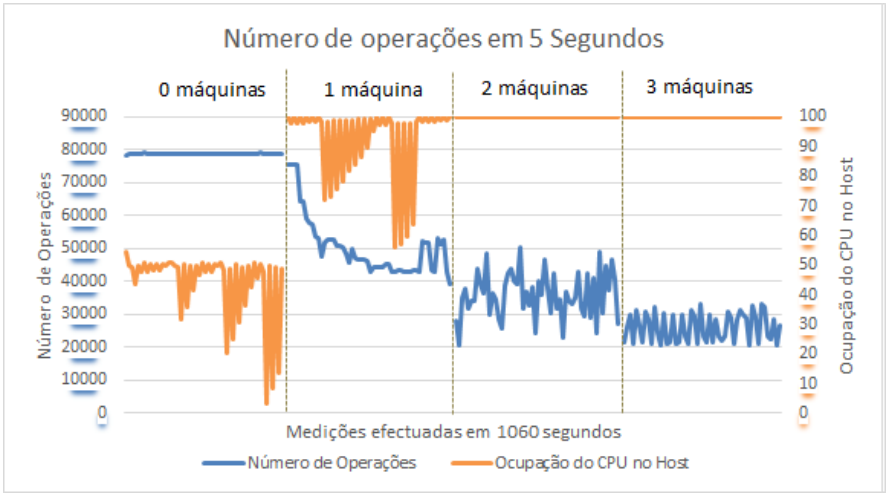


(d) Número médio de Operações efetuadas vs Média da ocupação do *CPU* no *Host*

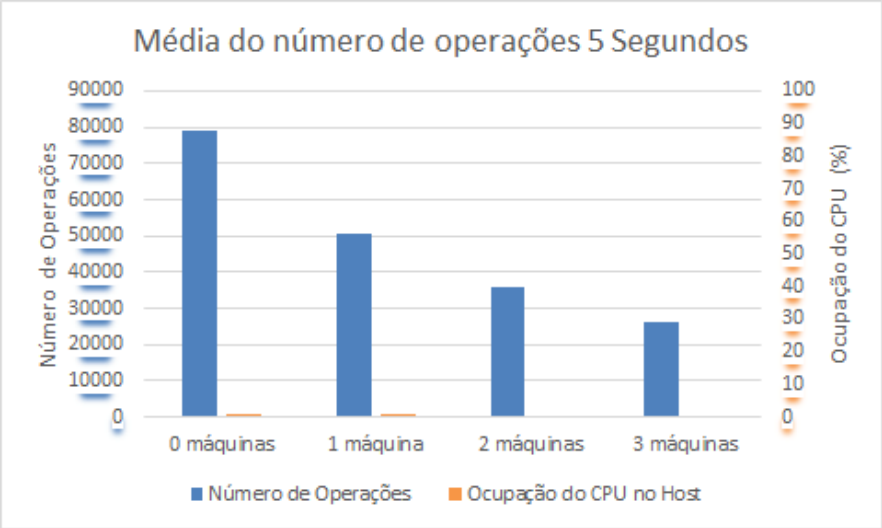
Figura D.1: Resultados obtidos com a monitorização com intervalo fixo de 2 segundos, variando a ocupação do *CPU*.



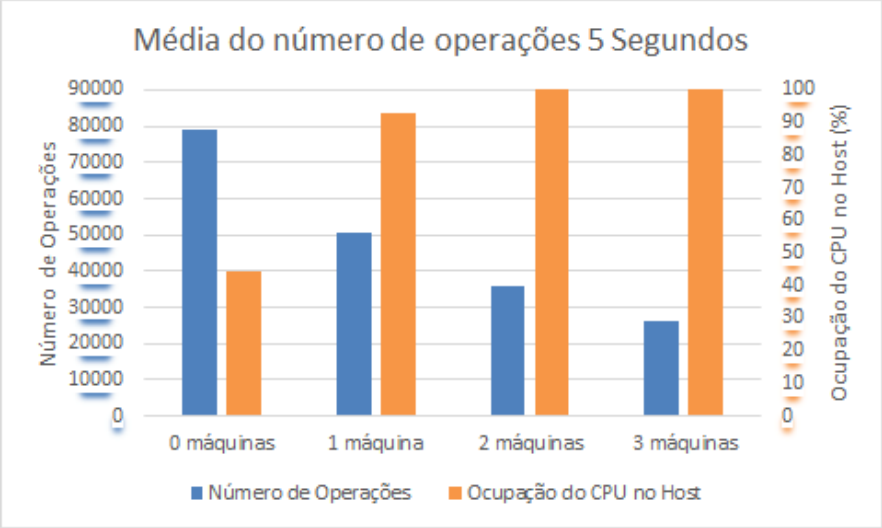
(a) Número de Operações efetuadas vs Medição de ocupação do *CPU* na máquina



(b) Número de Operações efetuadas vs Medição de ocupação do *CPU* no *Host*

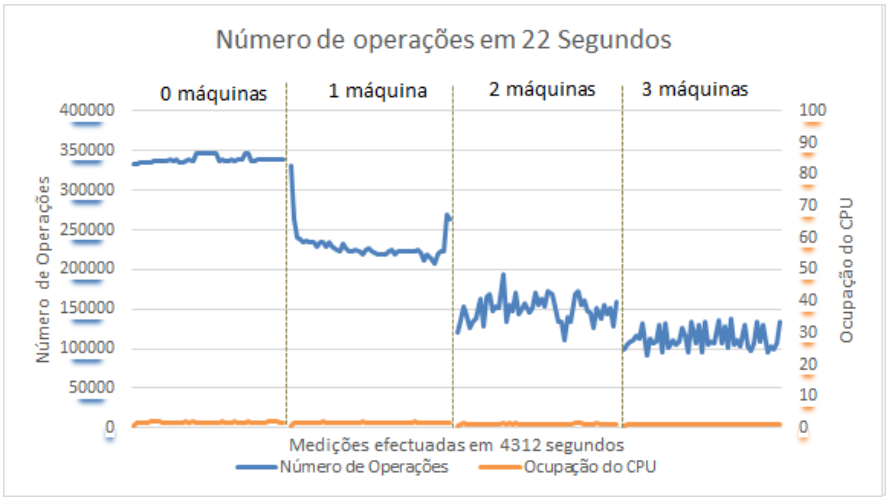


(c) Número médio de Operações efetuadas vs Média da ocupação do *CPU* na máquina

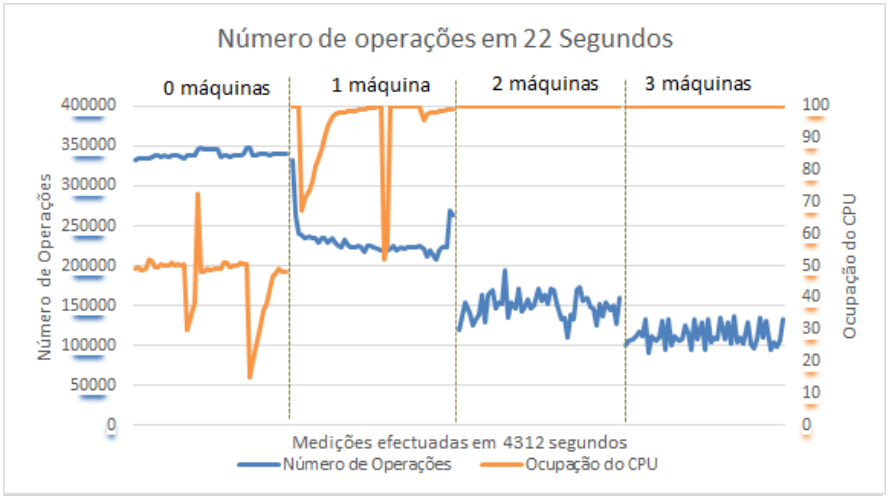


(d) Número médio de Operações efetuadas vs Média da ocupação do *CPU* no *Host*

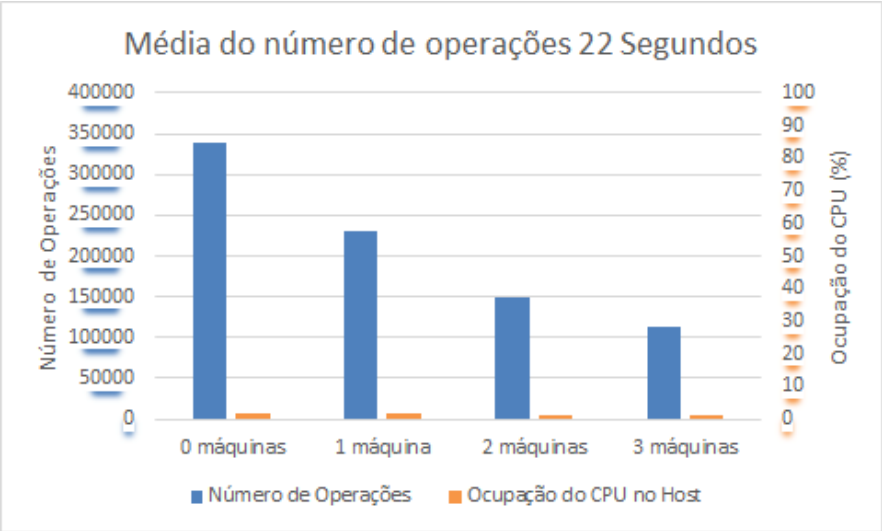
Figura D.2: Resultados obtidos com a monitorização com intervalo fixo de 5 segundos, variando a ocupação do *CPU*.



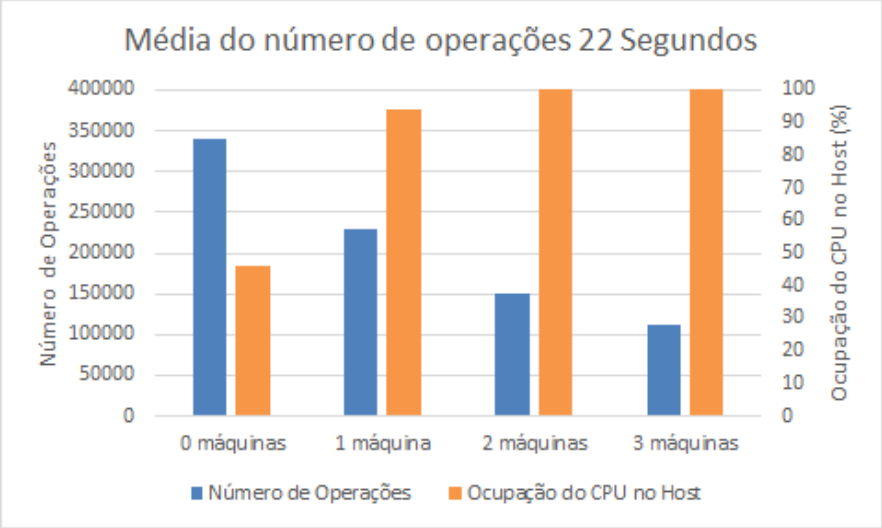
(a) Número de Operações efetuadas vs Medição de ocupação do *CPU* na máquina



(b) Número de Operações efetuadas vs Medição de ocupação do *CPU* no *Host*



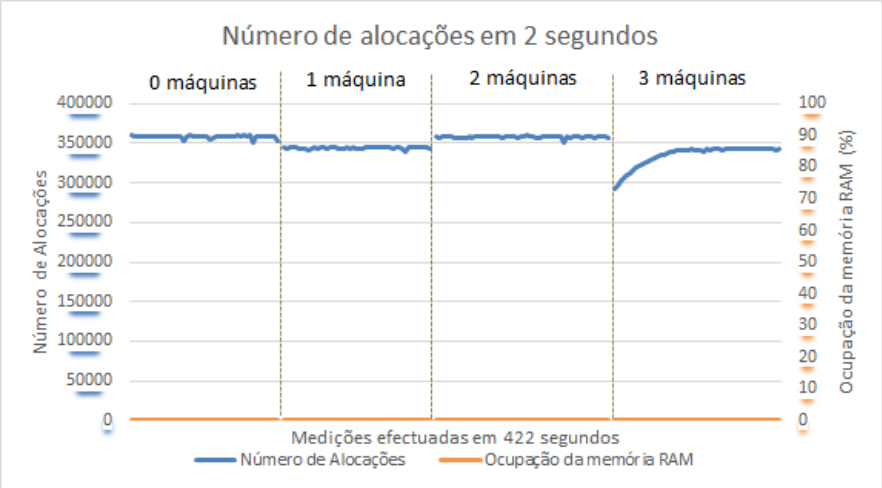
(c) Número médio de Operações efetuadas vs Média da ocupação do *CPU* na máquina



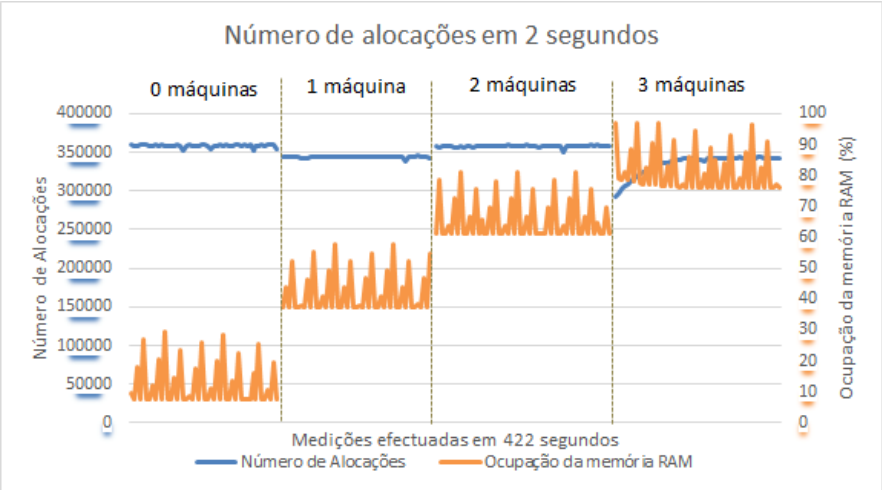
(d) Número médio de Operações efetuadas vs Média da ocupação do *CPU* no *Host*

Figura D.3: Resultados obtidos com a monitorização com intervalo fixo de 22 segundos, variando a ocupação do *CPU*.

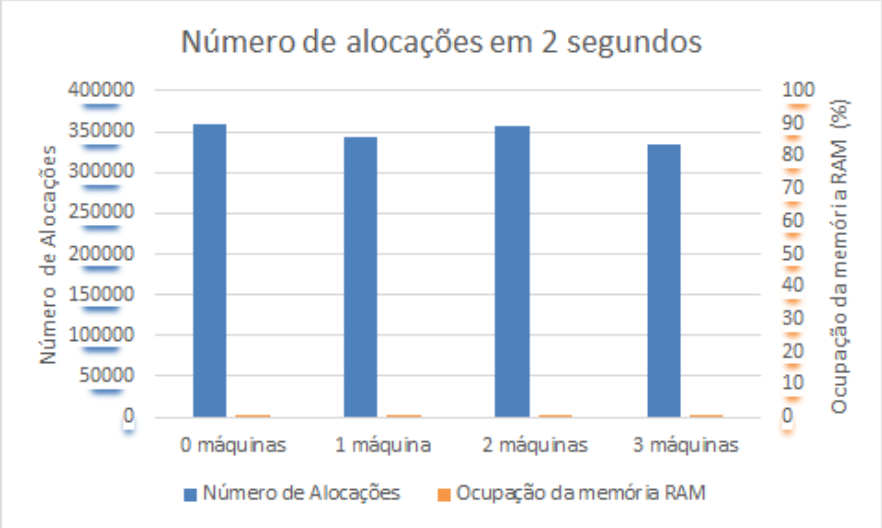
E | Resultados Completos dos Testes de Monitorização da Memória RAM (OpenVZ)



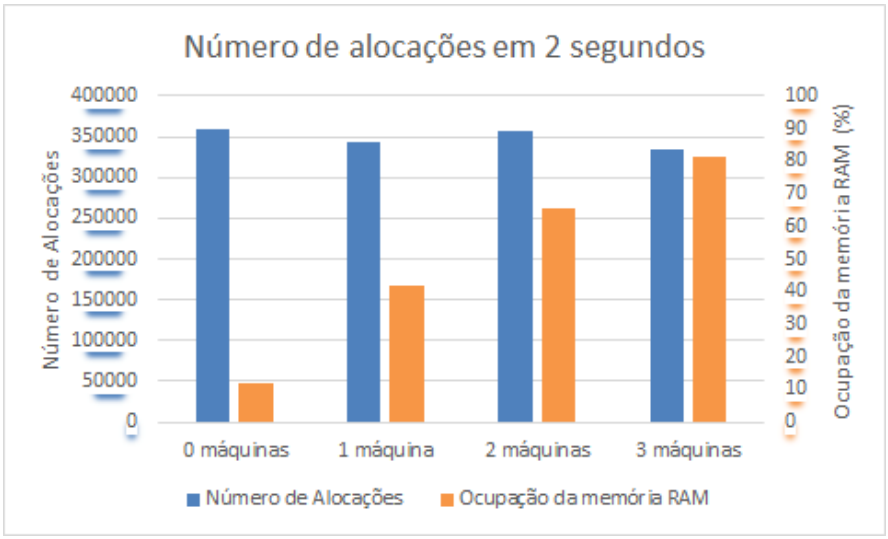
(a) Número de Operações efetuadas vs Medição de ocupação da memória *RAM* na máquina



(b) Número de Operações efetuadas vs Medição de ocupação da memória *RAM* no *Host*

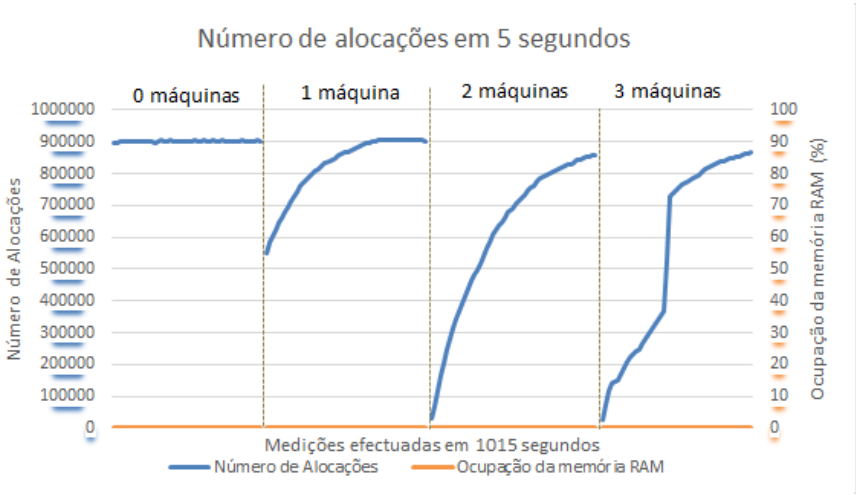


(c) Número médio de Operações efetuadas vs Média da ocupação da memória *RAM* na máquina

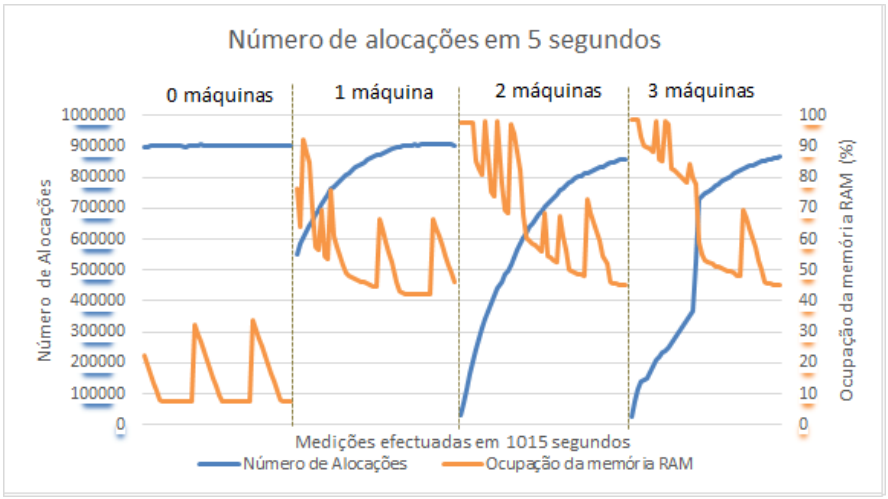


(d) Número médio de Operações efetuadas vs Média da ocupação da memória *RAM* no *Host*

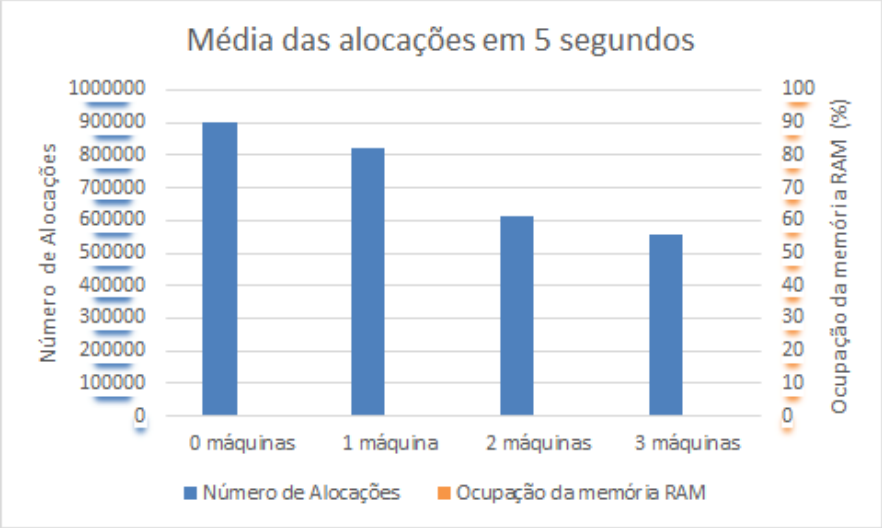
Figura E.1: Resultados obtidos com a monitorização com intervalo fixo de 2 segundos, variando a ocupação da memória *RAM*.



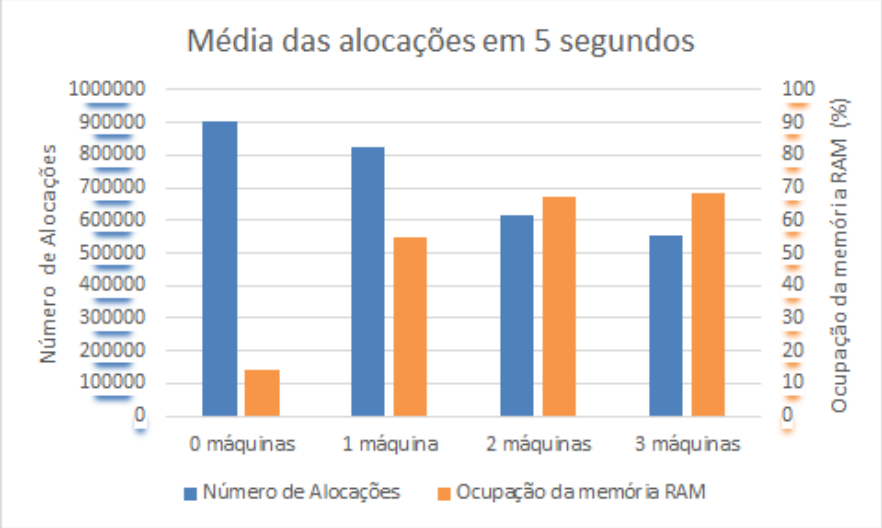
(a) Número de Operações efetuadas vs Medição de ocupação da memória *RAM* na máquina



(b) Número de Operações efetuadas vs Medição de ocupação da memória *RAM* no *Host*



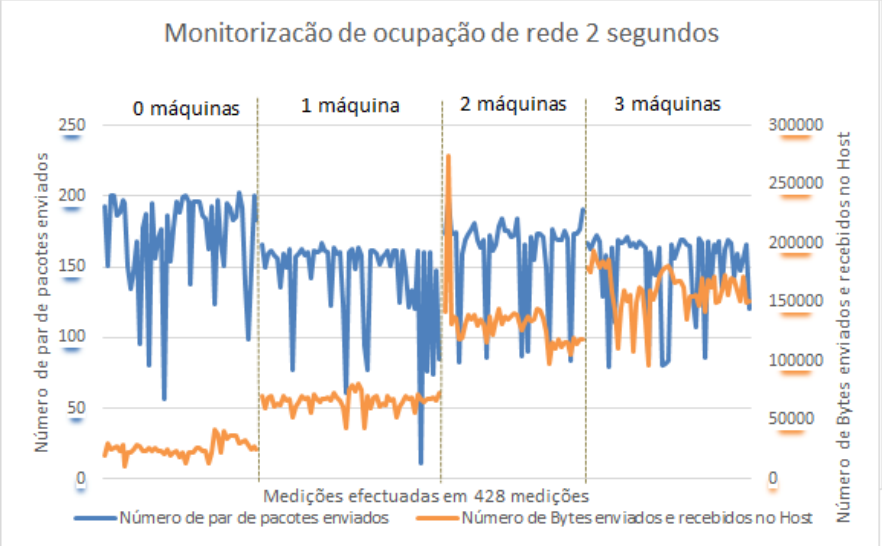
(c) Número médio de Operações efetuadas vs Média da ocupação da memória *RAM* na máquina



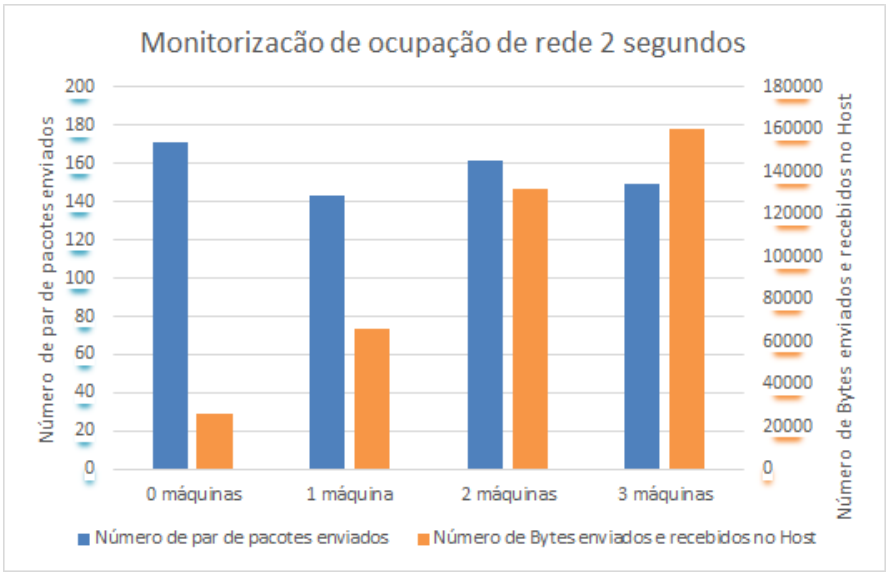
(d) Número médio de Operações efetuadas vs Média da ocupação da memória *RAM* no *Host*

Figura E.2: Resultados obtidos com a monitorização com intervalo fixo de 5 segundos, variando a ocupação da memória *RAM*.

F | Resultados Completos dos Testes de Monitorização da rede (OpenVZ)

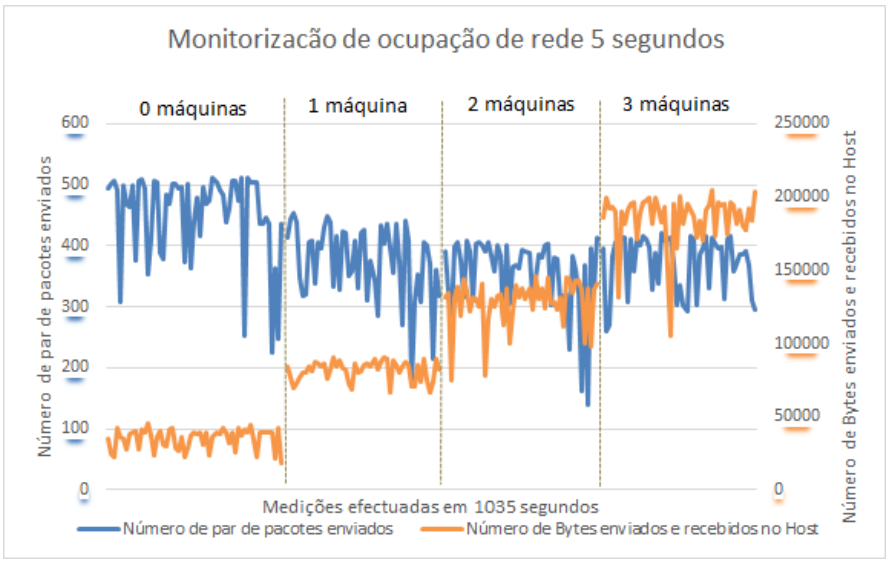


(a) Número de par de pacotes enviados vs Número de Bytes enviados e recebidos no *Host*

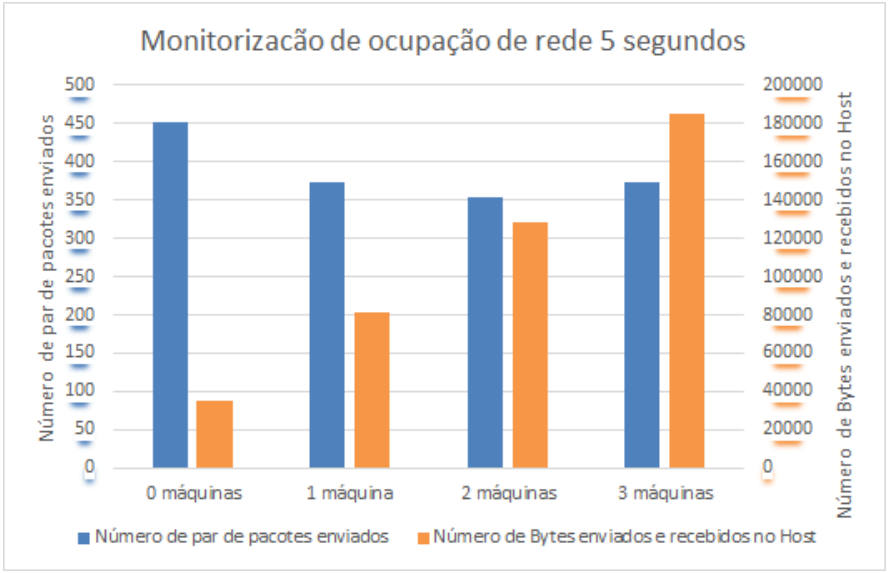


(b) Média do número de par de pacotes enviados vs Média número de Bytes enviados e recebidos no *Host*

Figura F.1: Resultados obtidos com a monitorização com intervalo fixo de 2 segundos, variando a ocupação da rede.

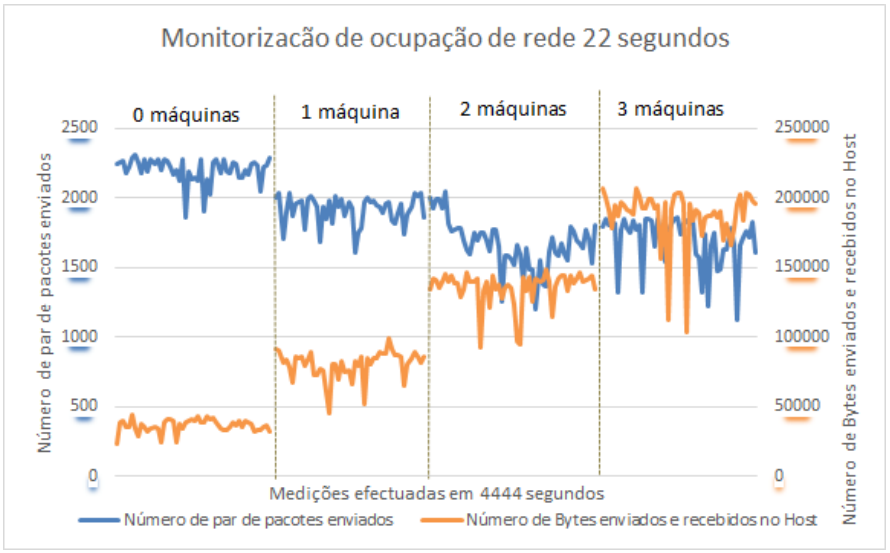


(a) Número de par de pacotes enviados vs Número de Bytes enviados e recebidos no *Host*

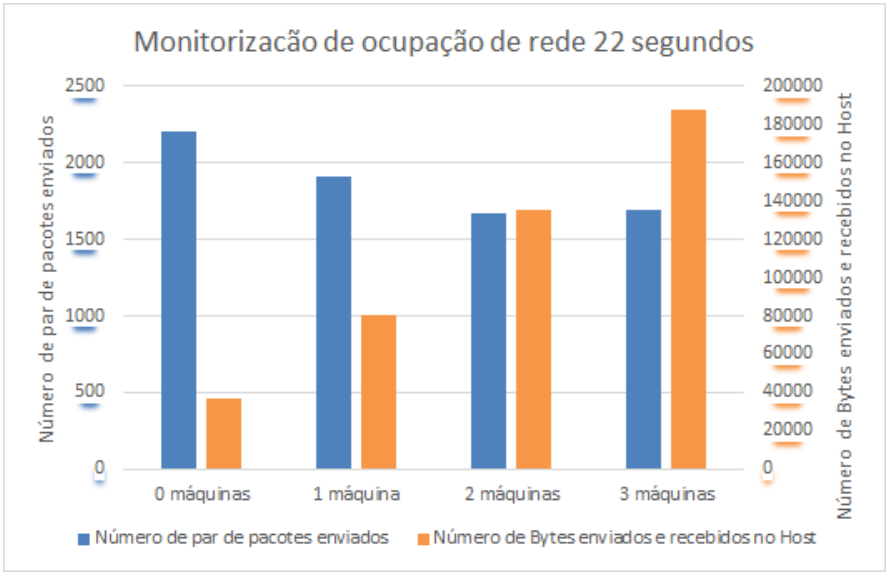


(b) Média do número de par de pacotes enviados vs Média número de Bytes enviados e recebidos no *Host*

Figura F.2: Resultados obtidos com a monitorização com intervalo fixo de 5 segundos, variando a ocupação da rede.



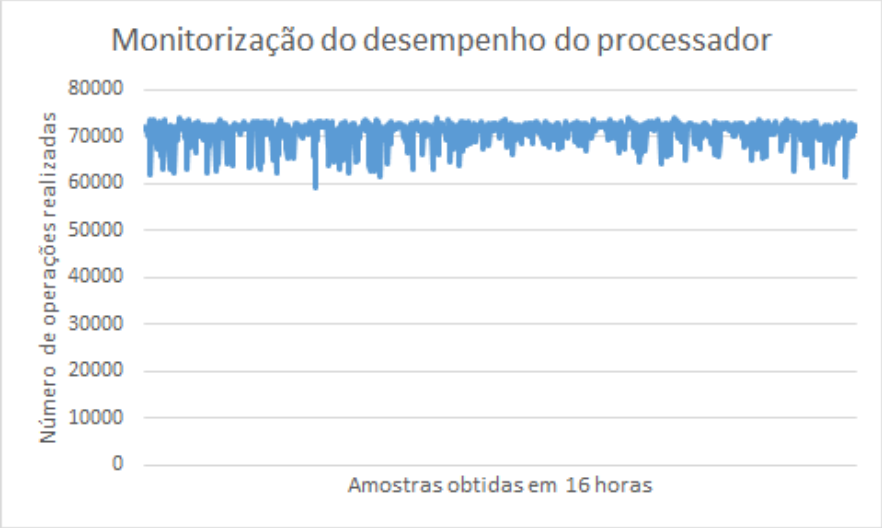
(a) Número de par de pacotes enviados vs Número de Bytes enviados e recebidos no *Host*



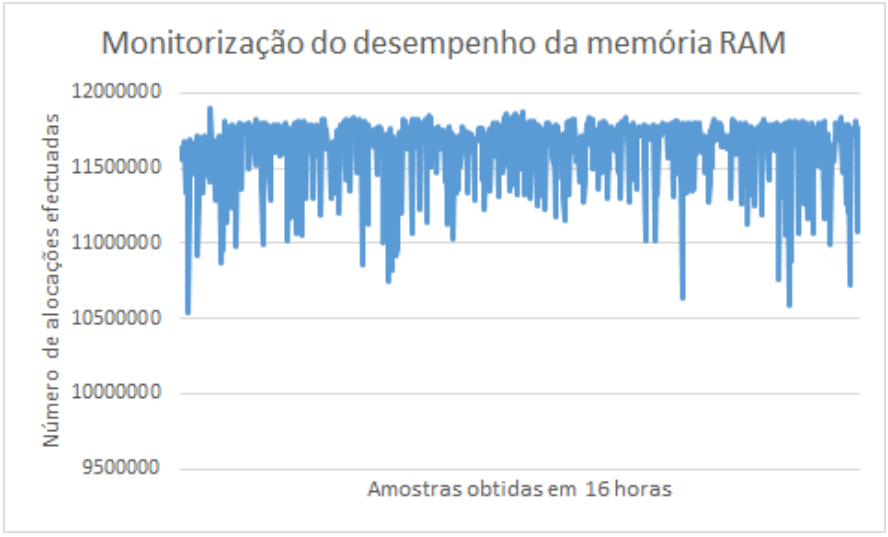
(b) Média do número de par de pacotes enviados vs Média número de Bytes enviados e recebidos no *Host*

Figura F.3: Resultados obtidos com a monitorização com intervalo fixo de 22 segundos, variando a ocupação da rede.

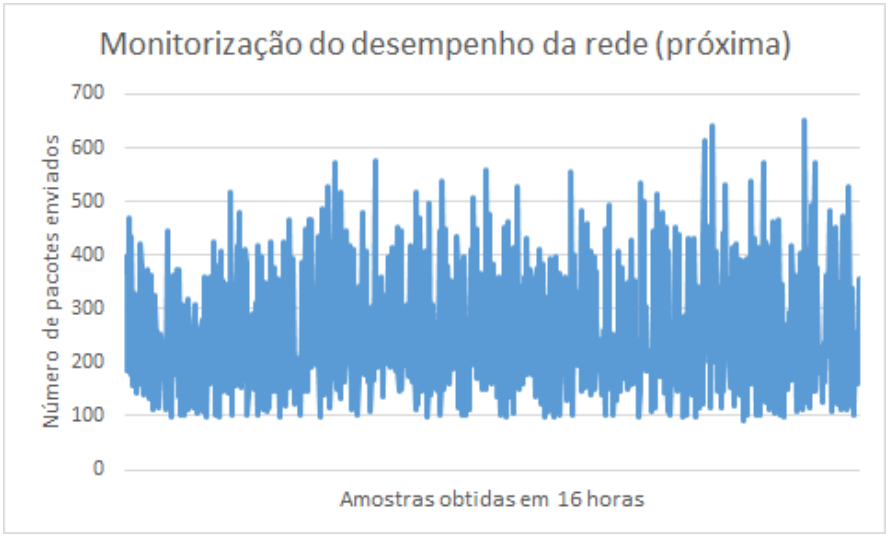
G | Valores obtidos pelos servidores de teste



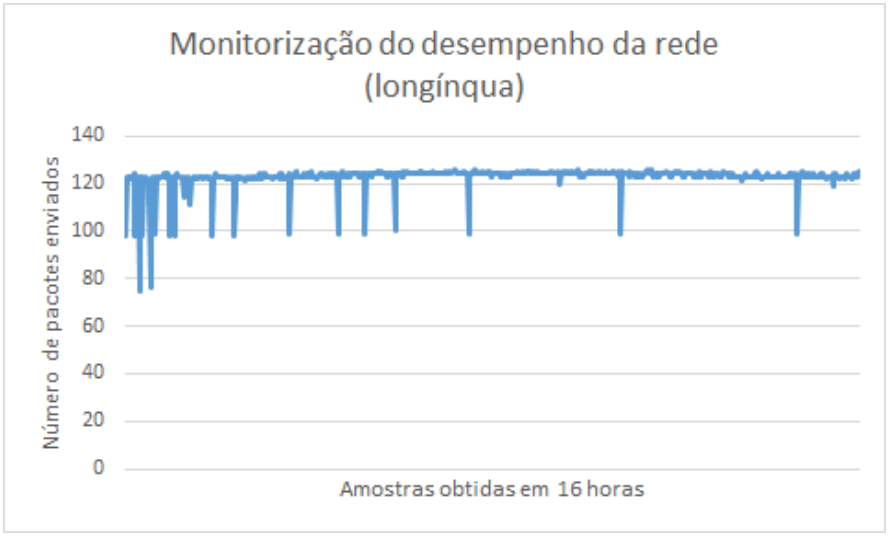
(a) Monitorização do processador da *probe* de *Londres*.



(b) Monitorização da memória *RAM* da *probe* de *Londres*.

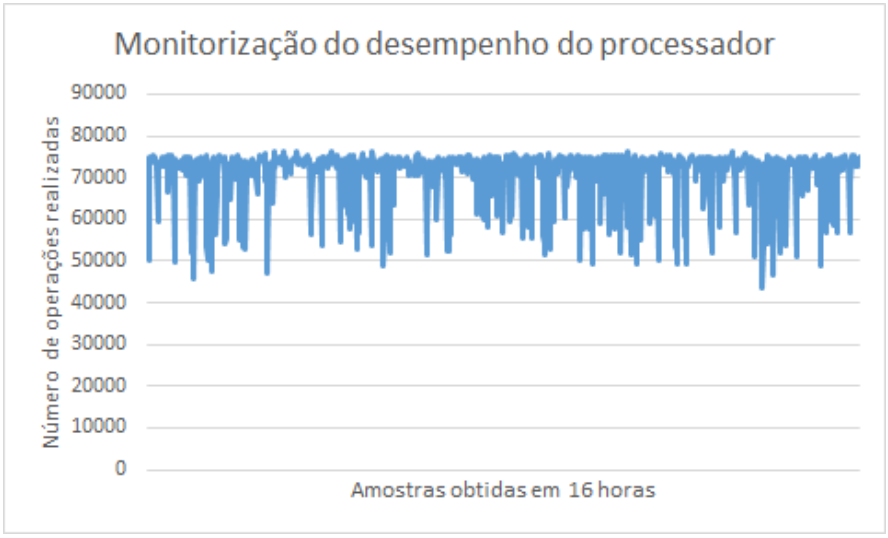


(c) Monitorização da placa de rede para redes próximas da *probe* de *Londres*.

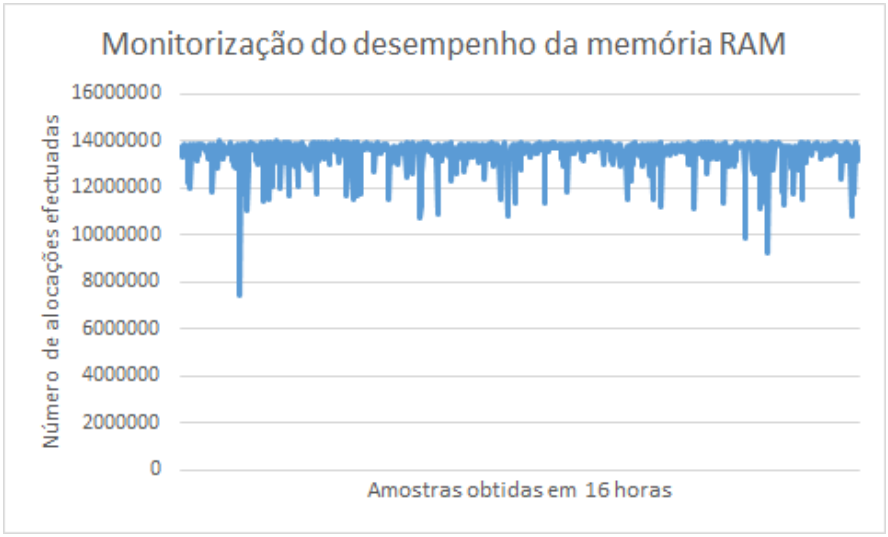


(d) Monitorização da placa de rede para redes longínquas da *probe* de *Londres*.

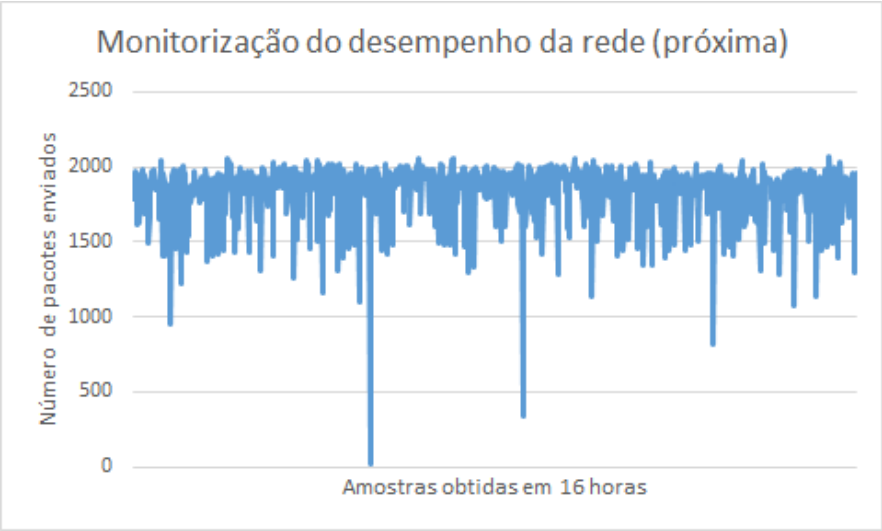
Figura G.1: Representações dos resultados da monitorização da *probe* de *Londres*.



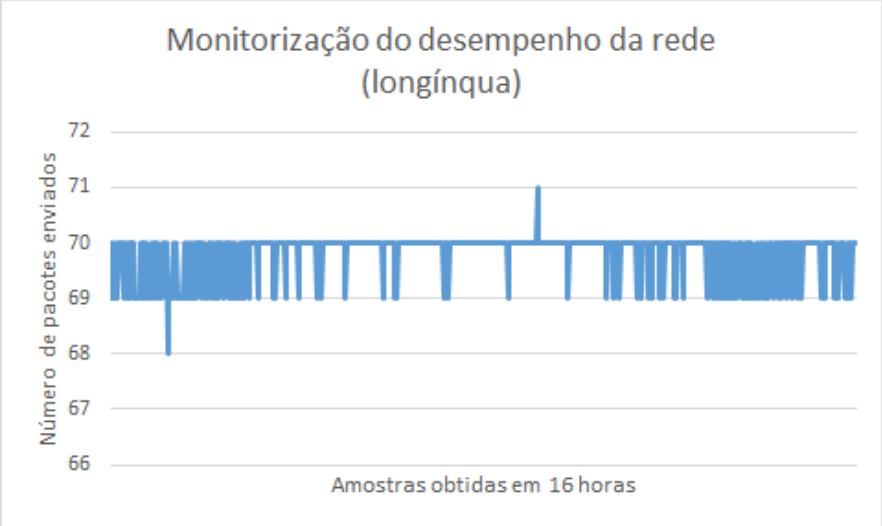
(a) Monitorização do processador da *probe* de *Suécia*.



(b) Monitorização da memória *RAM* da *probe* de *Suécia*.

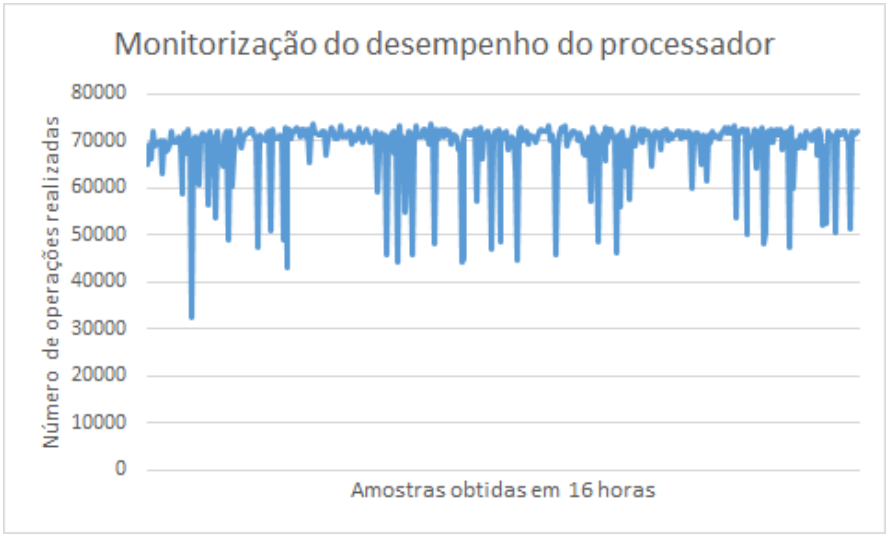


(c) Monitorização da placa de rede para redes próximas da *probe* de *Suécia*.

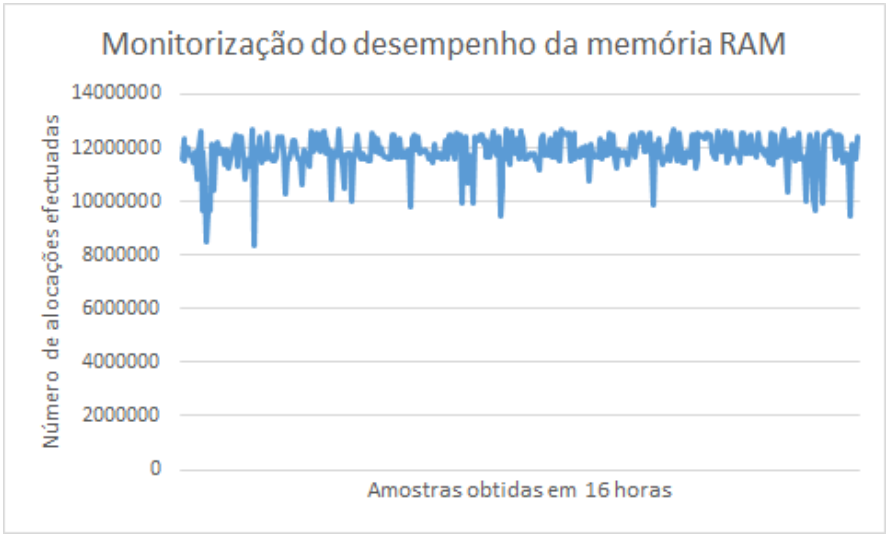


(d) Monitorização da placa de rede para redes longínquas da *probe* de *Suécia*.

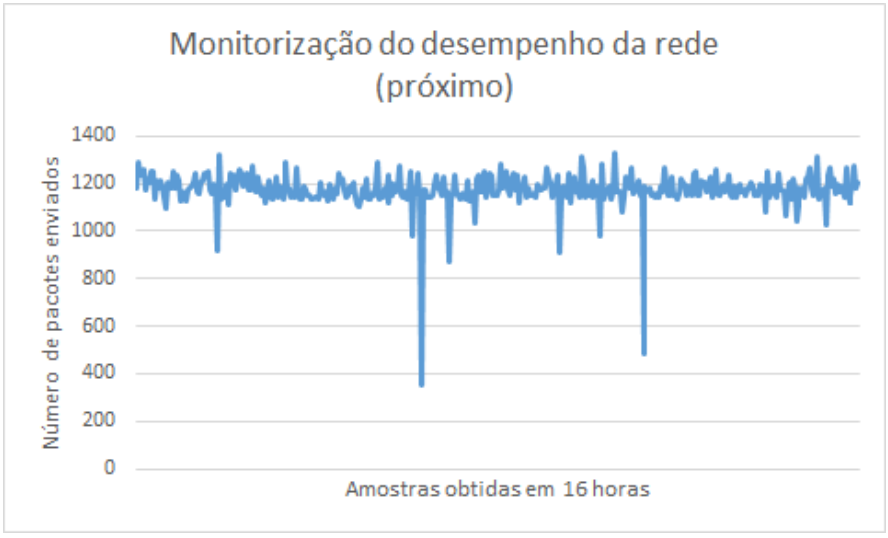
Figura G.2: Representações dos resultados da monitorização da *probe* de *Suécia*.



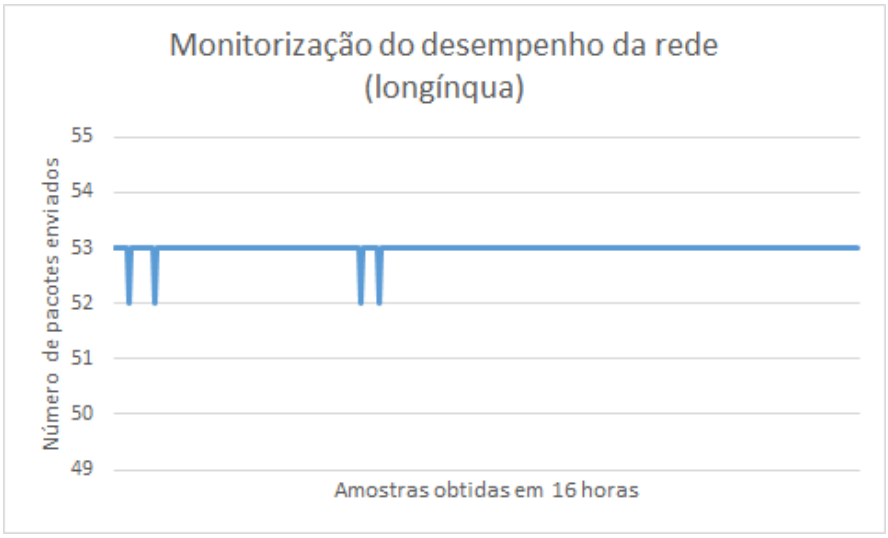
(a) Monitorização do processador da *probe* de *Moscovo*.



(b) Monitorização da memória *RAM* da *probe* de *Moscovo*.

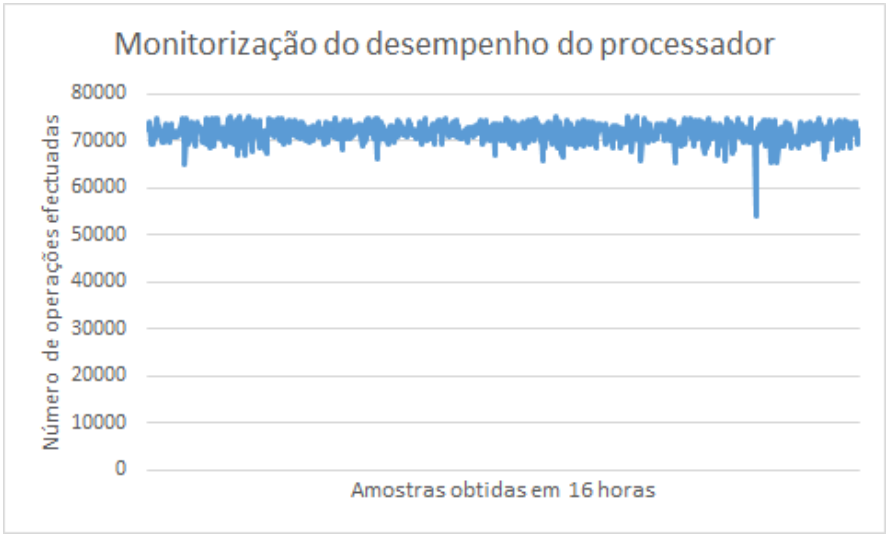


(c) Monitorização da placa de rede para redes próximas da *probe* de *Moscovo*.

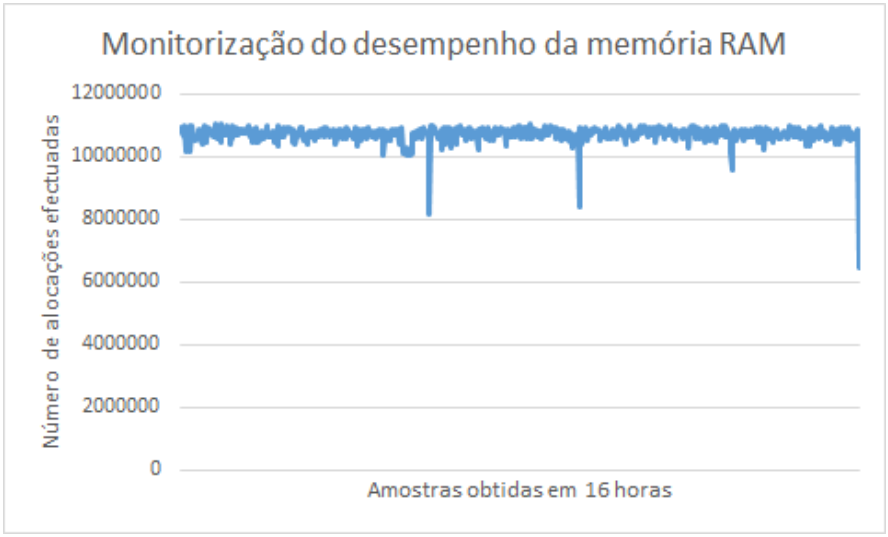


(d) Monitorização da placa de rede para redes longínquas da *probe* de *Moscovo*.

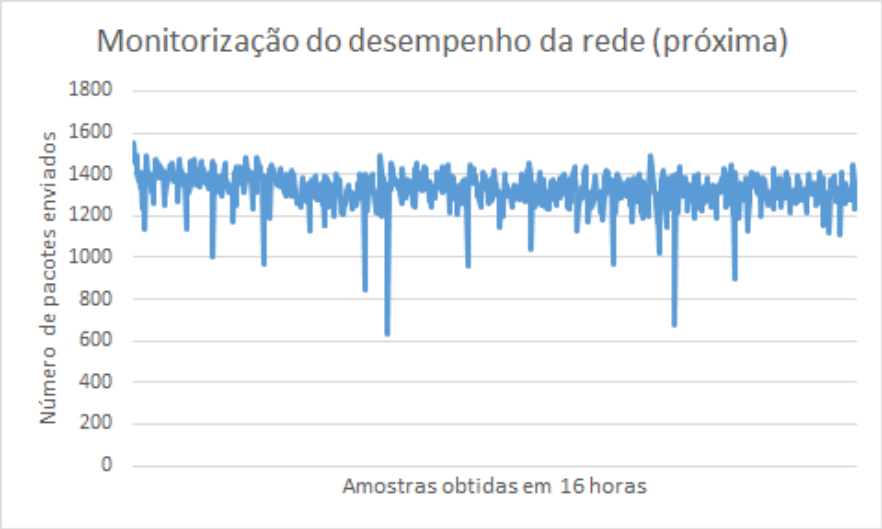
Figura G.3: Representações dos resultados da monitorização da *probe* de *Moscovo*.



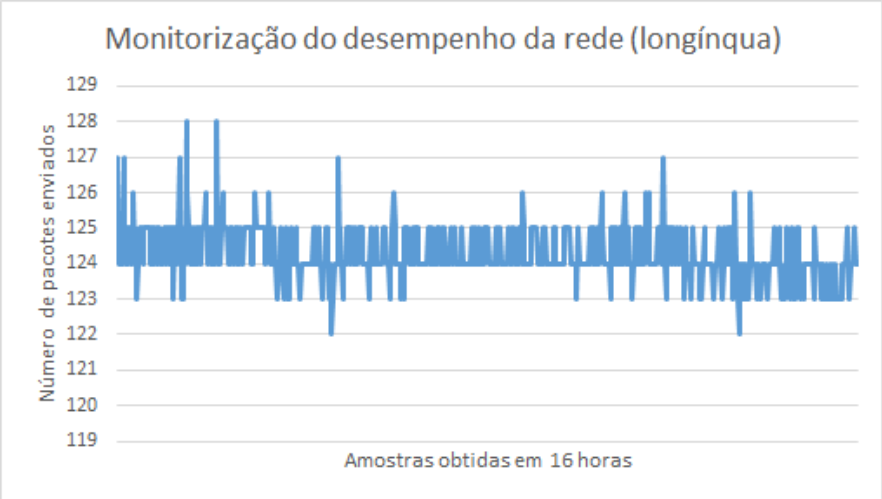
(a) Monitorização do processador da *probe* de *Amesterdão*.



(b) Monitorização da memória *RAM* da *probe* de *Amesterdão*.

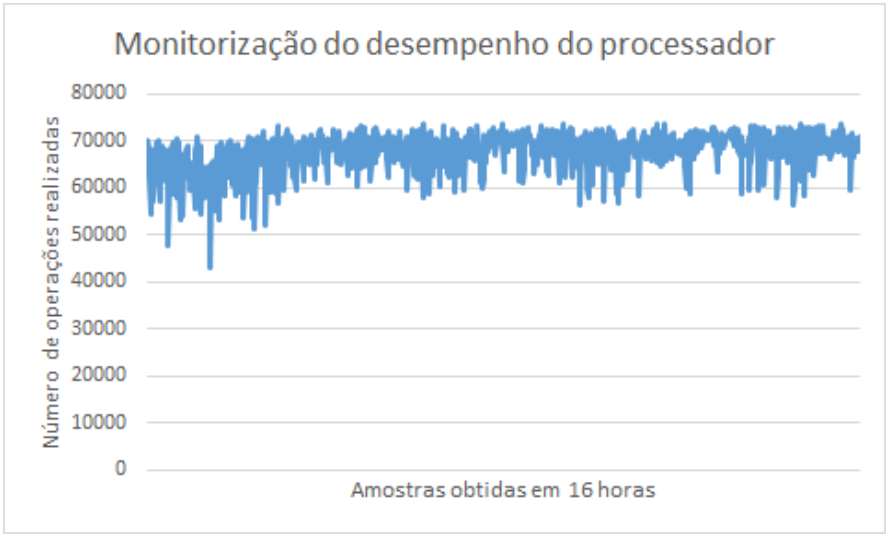


(c) Monitorização da placa de rede para redes próximas da *probe* de *Amesterdão*.

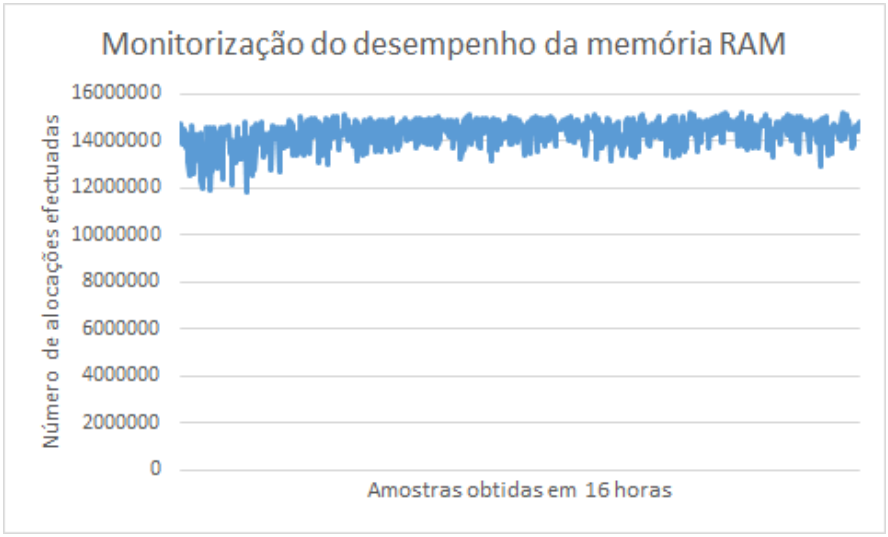


(d) Monitorização da placa de rede para redes longínquas da *probe* de *Amesterdão*.

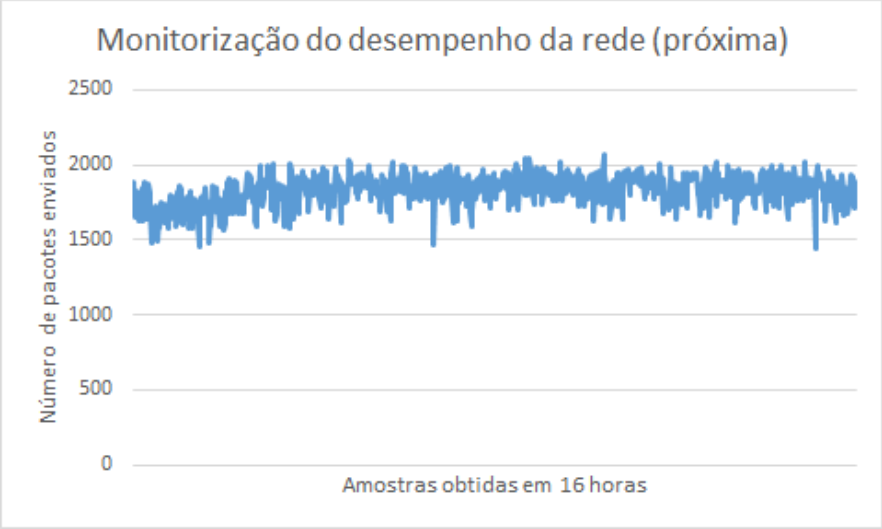
Figura G.4: Representações dos resultados da monitorização da *probe* de *Amesterdão*.



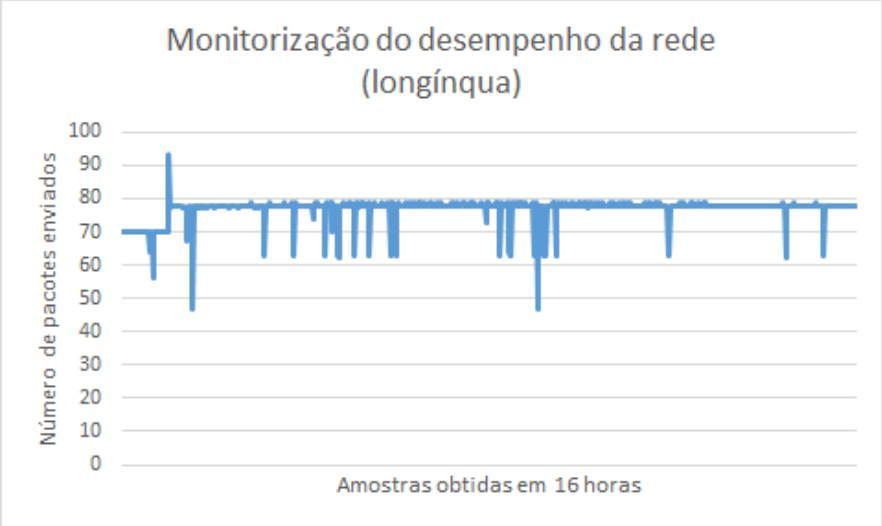
(a) Monitorização do processador da *probe* de *Milão*.



(b) Monitorização da memória *RAM* da *probe* de *Milão*.

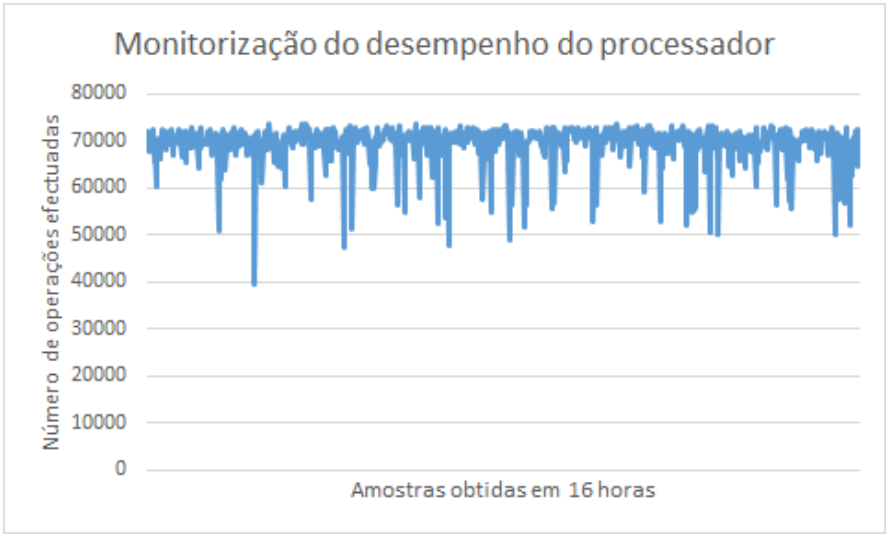


(c) Monitorização da placa de rede para redes próximas da *probe* de *Milão*.

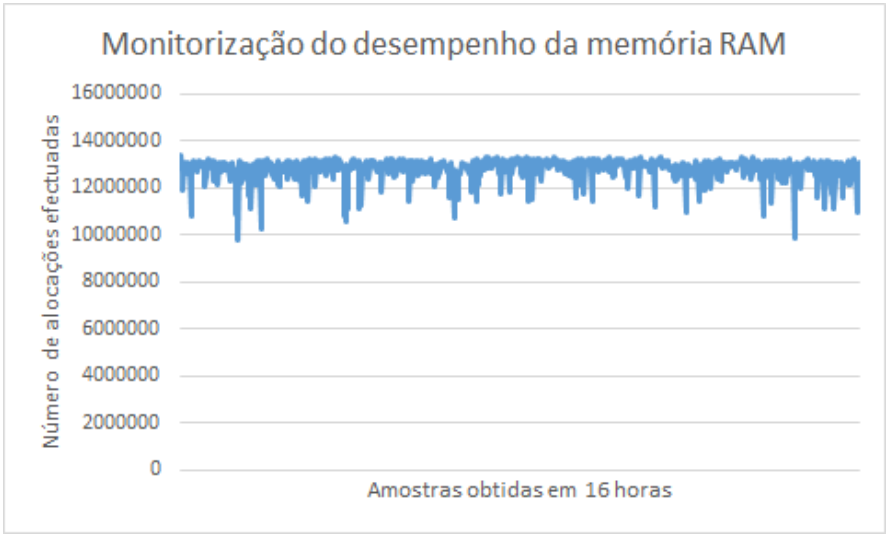


(d) Monitorização da placa de rede para redes longínquas da *probe* de *Milão*.

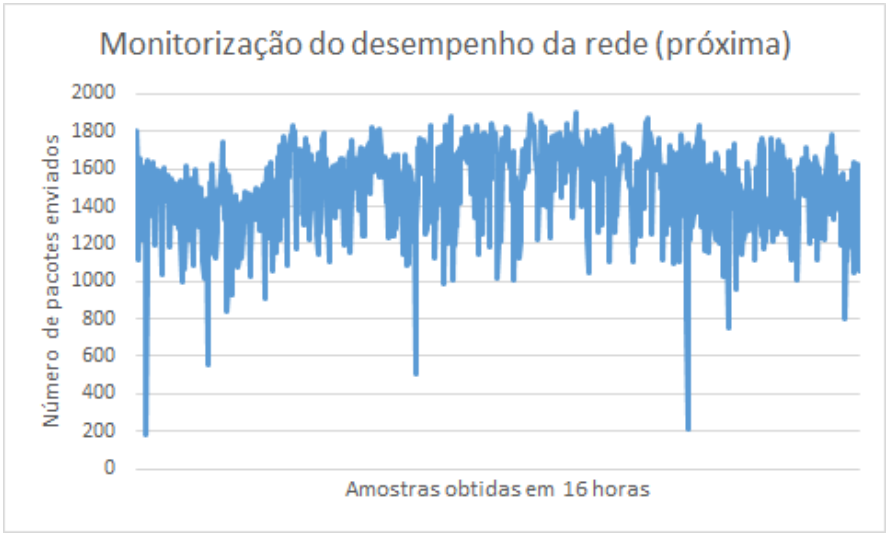
Figura G.5: Representações dos resultados da monitorização da *probe* de *Milão*.



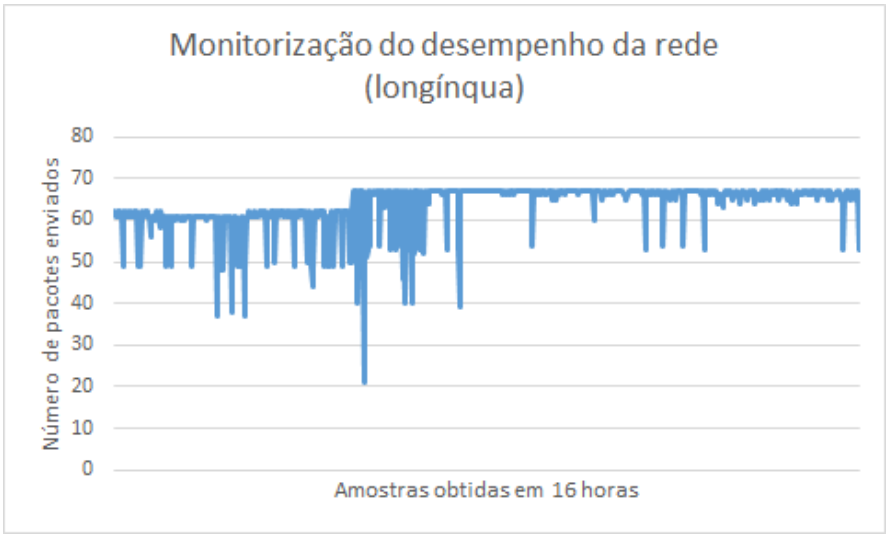
(a) Monitorização do processador da *probe* de *Islândia*.



(b) Monitorização da memória *RAM* da *probe* de *Islândia*.

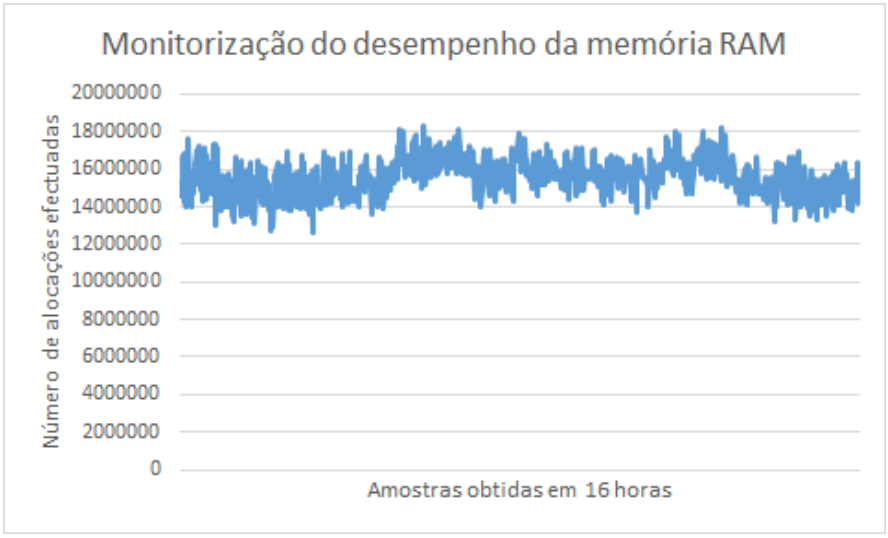


(c) Monitorização da placa de rede para redes próximas da *probe* de *Islândia*.

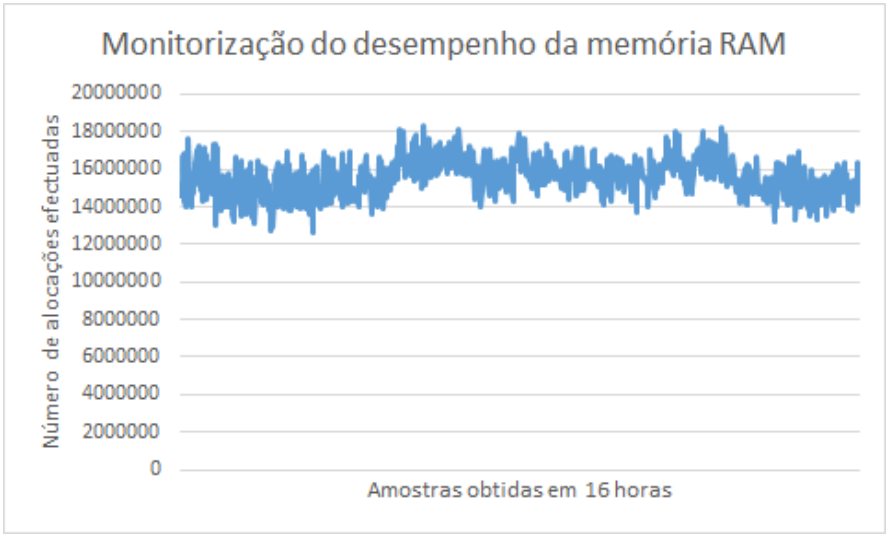


(d) Monitorização da placa de rede para redes longínquas da *probe* de *Islândia*.

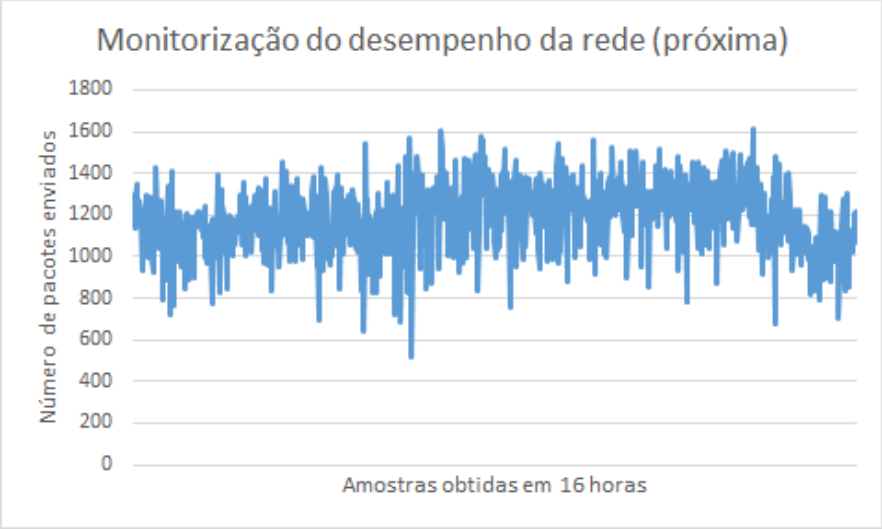
Figura G.6: Representações dos resultados da monitorização da *probe* de *Islândia*.



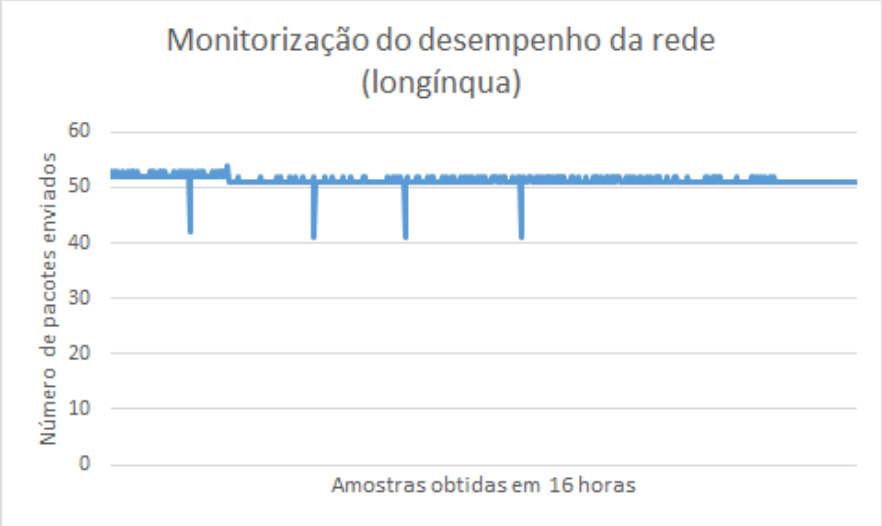
(a) Monitorização do processador da *probe* de *Israel*.



(b) Monitorização da memória *RAM* da *probe* de *Israel*.

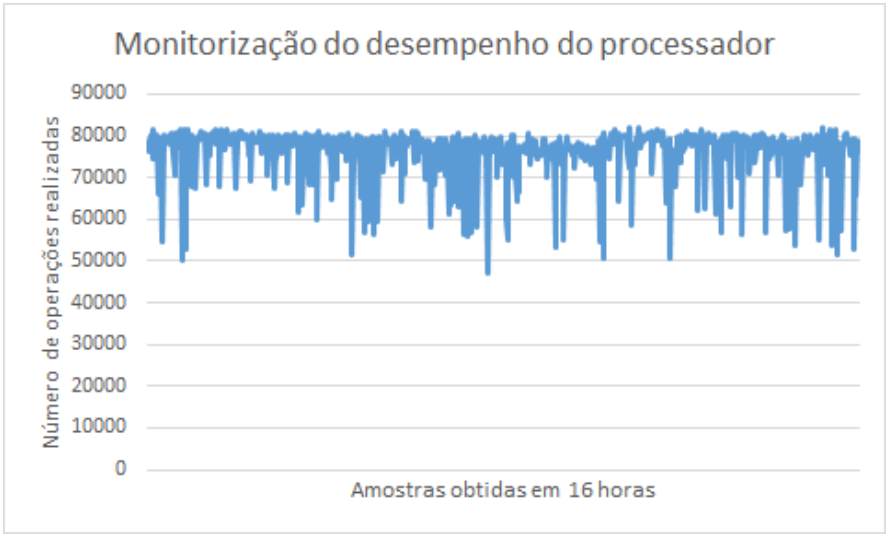


(c) Monitorização da placa de rede para redes próximas da *probe* de *Israel*.

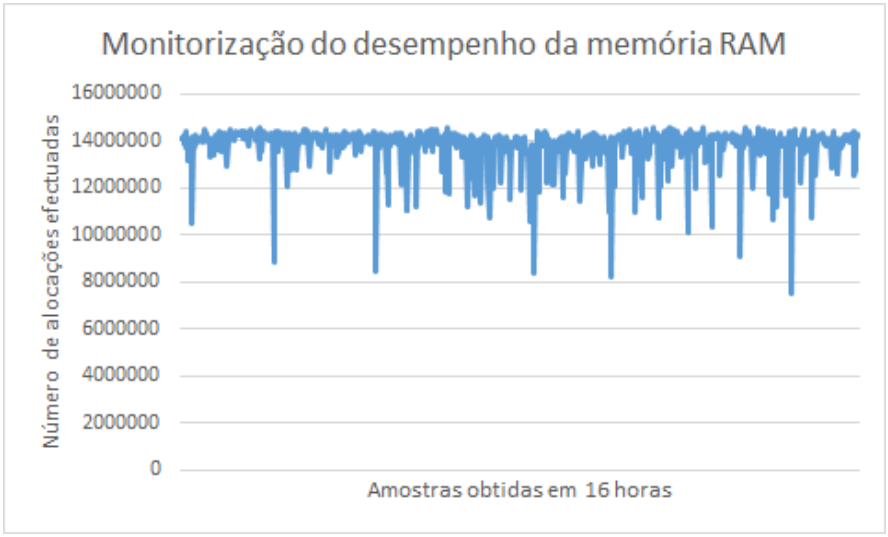


(d) Monitorização da placa de rede para redes longínquas da *probe* de *Israel*.

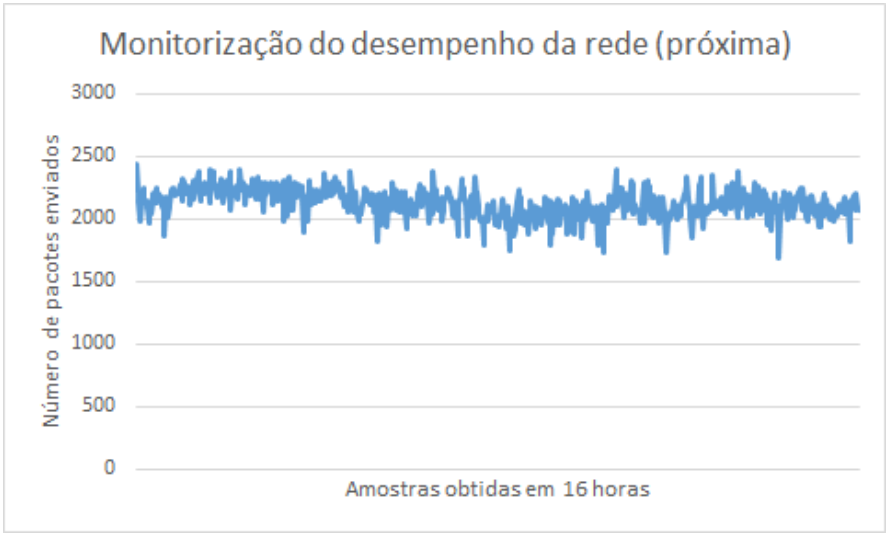
Figura G.7: Representações dos resultados da monitorização da *probe* de *Israel*.



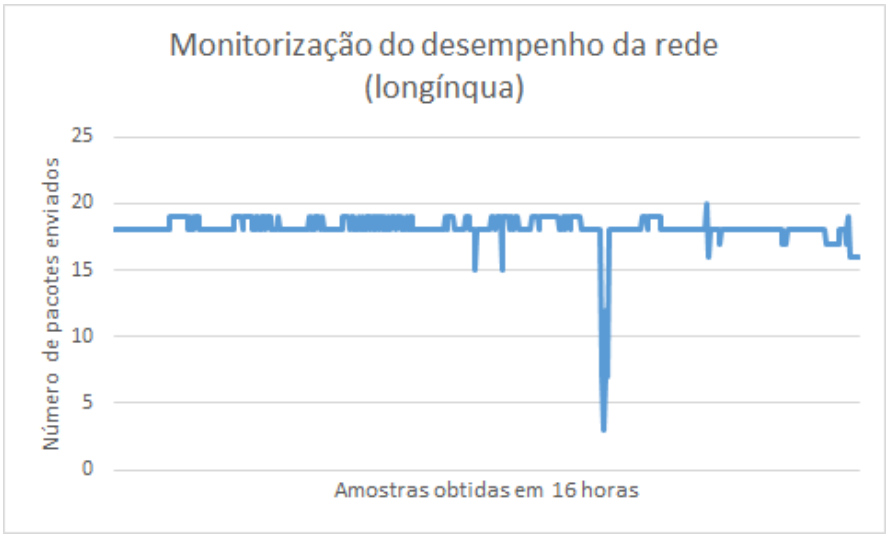
(a) Monitorização do processador da *probe* de *Hong Kong*.



(b) Monitorização da memória *RAM* da *probe* de *Hong Kong*.

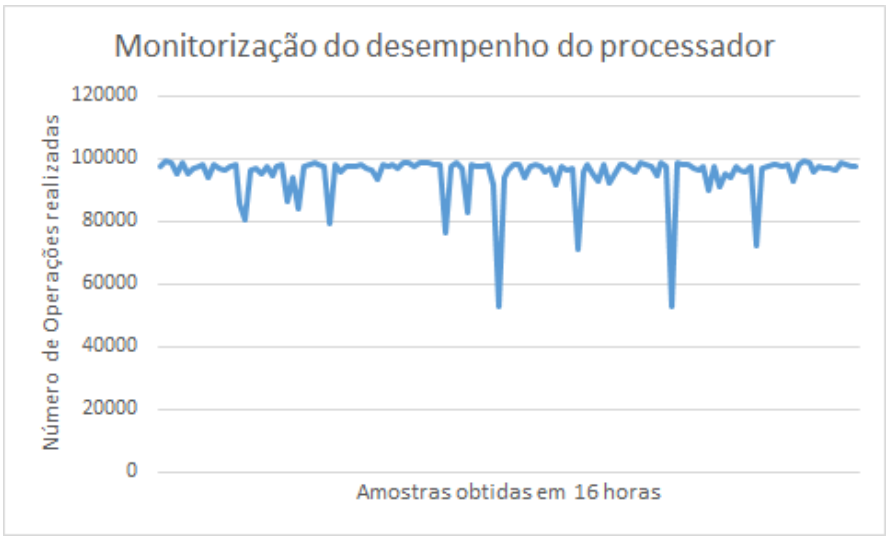


(c) Monitorização da placa de rede para redes próximas da *probe* de *Hong Kong*.

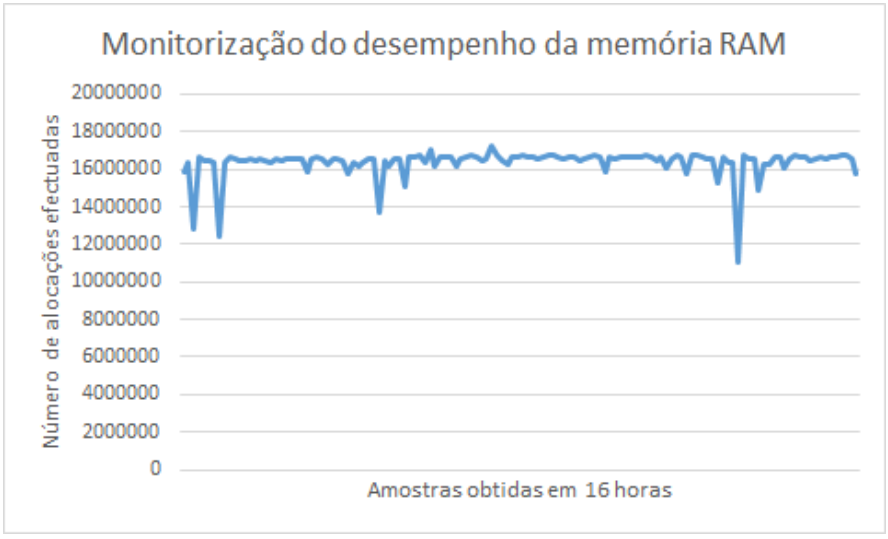


(d) Monitorização da placa de rede para redes longínquas da *probe* de *Hong Kong*.

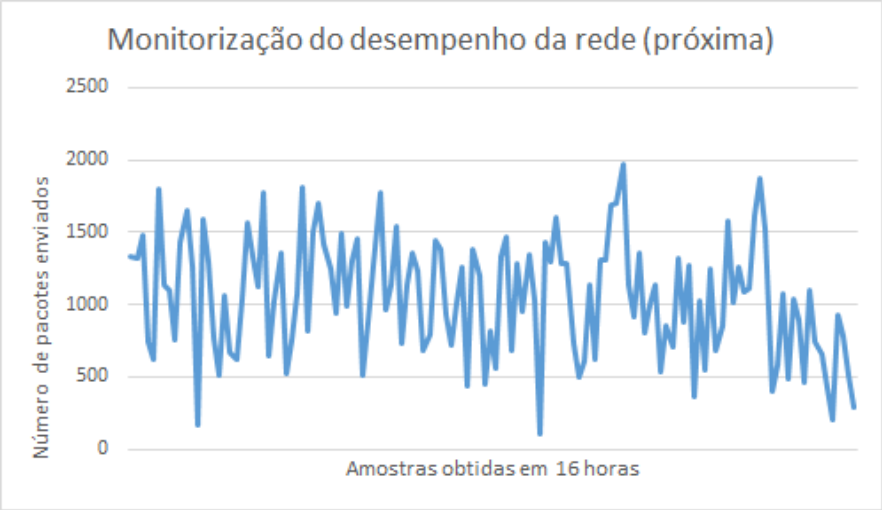
Figura G.8: Representações dos resultados da monitorização da *probe* de *Hong Kong*.



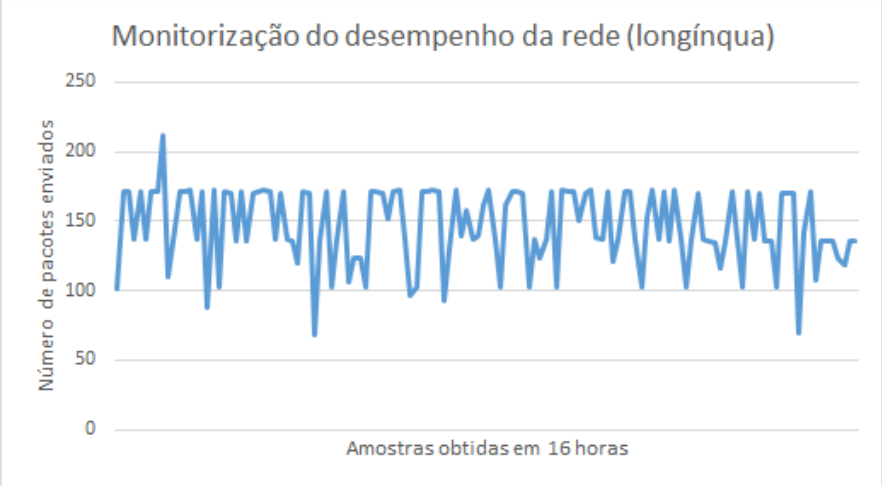
(a) Monitorização do processador da *probe* de *Madrid*.



(b) Monitorização da memória *RAM* da *probe* de *Madrid*.

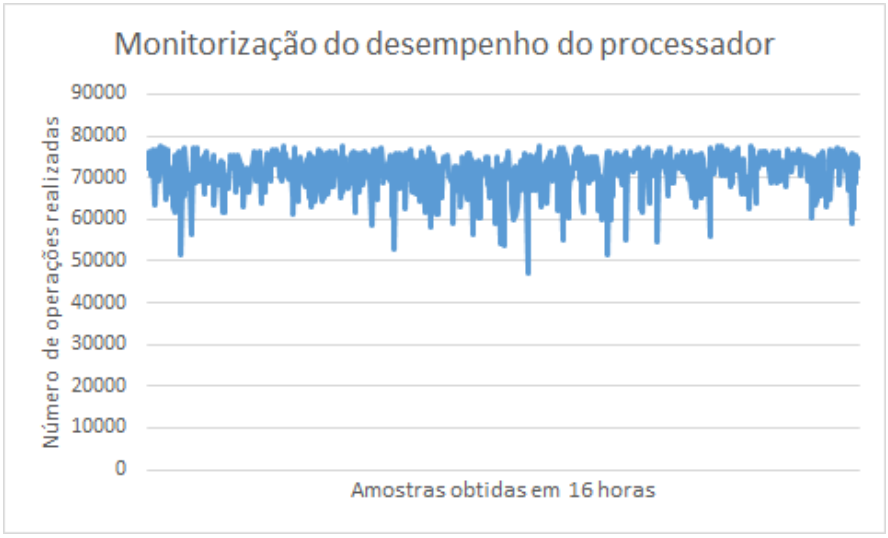


(c) Monitorização da placa de rede para redes próximas da *probe* de *Madrid*.

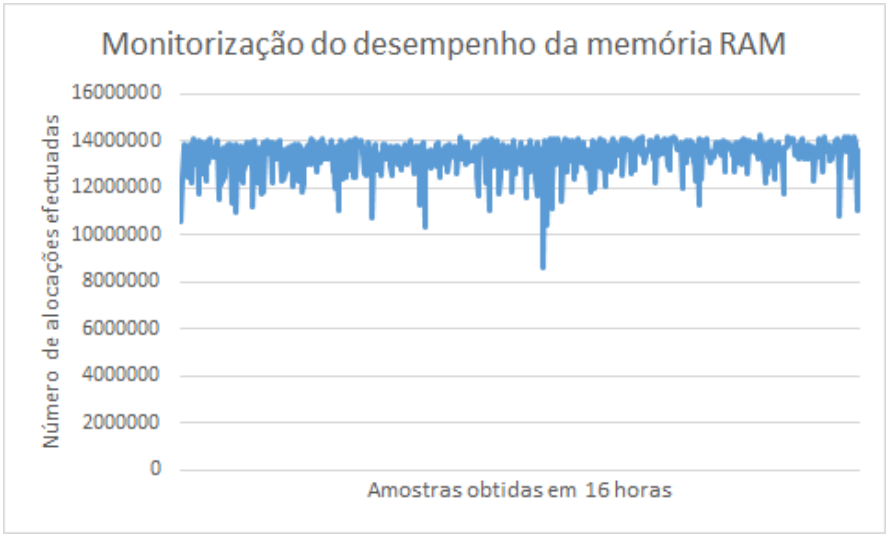


(d) Monitorização da placa de rede para redes longínquas da *probe* de *Madrid*.

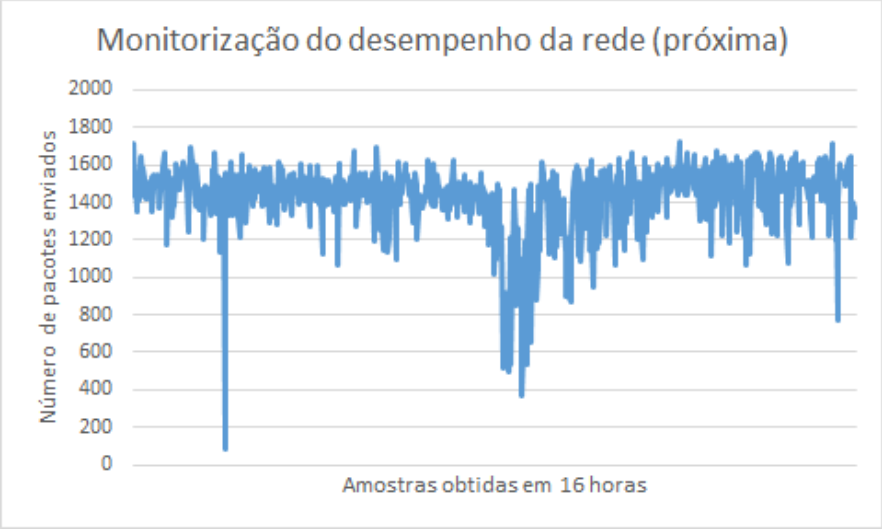
Figura G.9: Representações dos resultados da monitorização da *probe* de *Madrid*.



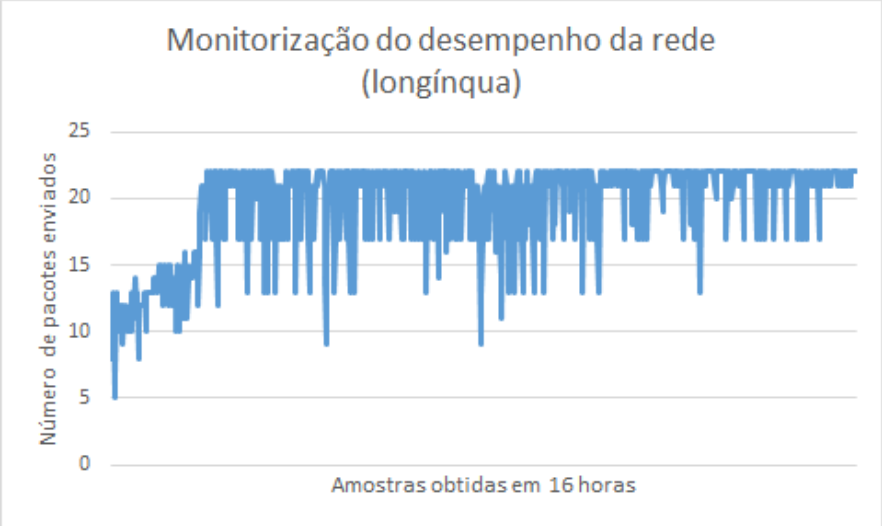
(a) Monitorização do processador da *probe* de *Chile*.



(b) Monitorização da memória *RAM* da *probe* de *Chile*.



(c) Monitorização da placa de rede para redes próximas da *probe* de *Chile*.



(d) Monitorização da placa de rede para redes longínquas da *probe* de *Chile*.

Figura G.10: Representações dos resultados da monitorização da *probe* de *Chile*.